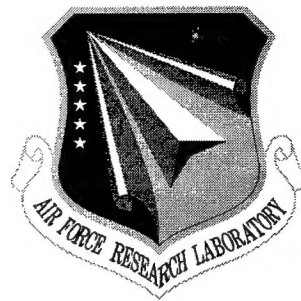AFRL-IF-RS-TR-2001-33
Final Technical Report
March 2001

ARTIFICIAL INTELLIGENCE AND OPERATIONS RESEARCH: CHALLENGES AND OPPORTUNITIES IN PLANNING AND SCHEDULING

Cornell University

Carla P. Gomes

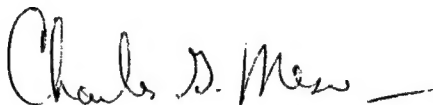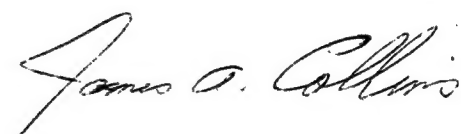*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*

20010507 075

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2001-33 has been reviewed and is approved for publication.

APPROVED: *[signature]*

CHARLES G. MESSENGER
Project Engineer

FOR THE DIRECTOR: *[signature]*

JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	MARCH 2001	Final Dec 97 - Nov 00

4. TITLE AND SUBTITLE
ARTIFICIAL INTELLIGENCE AND OPERATIONS RESEARCH: CHALLENGES AND OPPORTUNITIES IN PLANNING AND SCHEDULING

5. FUNDING NUMBERS
C - F30602-98-1-0008
PE - 61102F
PR - 2304
TA - A1
WU - P1

6. AUTHOR(S)
Carla P. Gomes

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Cornell University
Computer Science Department
Ithaca NY 14853

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFTD
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2001-33

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: Charles G. Messenger/IFTD/(315) 330-3528

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
The objective of this effort was to research technology, tools, and techniques to support more efficient techniques for solving hard computational problems. Both the Artificial Intelligence (AI) community and the Operations Research (OR) community are interested in developing techniques for solving hard combinatorial problems, in particular in the domain of planning and scheduling. AI approaches encompass a rich collection of knowledge representation formalisms for dealing with a wide variety of real-world problems. OR based techniques have demonstrated the ability to identify optimal and locally optimal solutions for well-defined problem spaces. In general, however, OR solutions are restricted to rigid models with limited expressive power. AI techniques, on the other hand, provide richer and more flexible representations of real-world problems, supporting efficient constraint-based reasoning mechanisms as well as mixed initiative frameworks, which allow the human expertise to be in the loop. The challenge lies in providing representations that are expensive enough to describe real-world problems and at the same time guaranteeing good and fast solutions.

14. SUBJECT TERMS
Artificial Intelligence (AI), Operations Research (OR), Algorithms

15. NUMBER OF PAGES
72

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

Artificial Intelligence and Operations Research: Challenges and Opportunities in Planning and Scheduling

Carla P. Gomes
Computer Science Department
Cornell University
Ithaca, NY 14853
gomes@cs.cornell.edu

1   Introduction

Both the Artificial Intelligence (AI) community and the Operations Research (OR) community are interested in developing techniques for solving hard combinatorial problems, in particular in the domain of planning and scheduling. AI approaches encompass a rich collection of knowledge representation formalisms for dealing with a wide variety of real-world problems. Some examples are constraint programming representations, logical formalisms, declarative and functional programming languages such as Prolog and Lisp, Bayesian models, rule-based formalism, etc. The downside of such rich representations is that in general they lead to intractable problems, and we therefore often cannot use such formalisms for handling realistic size problems. OR, on the other hand, has focused on more tractable representations, such as linear programming formulations. OR based techniques have demonstrated the ability to identify optimal and locally optimal solutions for well-defined problem spaces. In general, however, OR solutions are restricted to rigid models with limited expressive power. AI techniques, on the other hand, provide richer and more flexible representations of real-world problems, supporting efficient constraint-based reasoning mechanisms as well as mixed initiative frameworks, which allow the human expertise to be in the loop. The challenge lies in providing representations that are expressive enough to describe real-world problems and at the same time guaranteeing good and fast solutions. Figure 1 provides a high-level view of our perspective of the integration of AI and OR.

Integration of Artificial Intelligence & Operations Research Techniques

AI

Representations
Constraint Languages
Logic Formalisms
Object-Oriented Prog.
Bayesian Nets
Rule Based Systems
. . .
Tools
Constraint Propagation
Systematic Search
Stochastic Search
. . .
Pros / Cons
Rich Representations
Computational Complexity

OR
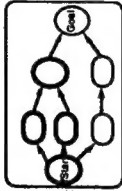
Representations
Mathematical
Modeling Languages
Linear & Non-linear
(In)Equalities
. . .
Tools
Linear Programming
Mixed-Integer Prog,
Non-linear Models
. . .
Pros / Cons
More Tractable (LP)
Primarily Complete Info
Limited Representations

Combinatorial Problems

Scheduling

Planning

THE CHALLENGE

UNIFY APPROACHES TO:

SCALE UP SOLUTIONS

EXPLOIT RANDOMIZATION

ANALYZE COMPLEXITY
(phase transition)

EXPLOIT PROBLEM STRUCTURE

INCREASE ROBUSTNESS

C.P.Gomes

FRAGILE

Below we present some of the main themes in OR followed by a discussion on several topics that, in our opinion, represent opportunities for integration of AI and OR techniques. As an appendix to this report we include a few papers that elaborate on some of the topics.

2 Main Themes in Operations Research

Optimization and Linear Programming

Traditionally, the Operations Research community has focused on solving optimization problems. Linear Programming plays a major role in OR methods. Work done by Leonid Kantorovich in 1939 is considered the main precursor to Linear Programming (LP). In 1947, George Dantzig developed LP and the simplex method, initially conceived to speed up the process of providing a time-staged deployment, training and logistical program for military applications.[1] Interestingly, the word "programming" in Linear Programming has nothing to do with computer programming, but rather with the notion of "program" as used by the military to refer to plans of military operations. The simplex method made it possible to consider larger problems in areas as diverse as transportation, production, resource allocation, and scheduling problems.

The main extensions of LP are Integer Programming (IP) and Mixed Integer Programming (MIP) and Stochastic programming. IP and MIP extend LP to deal with integrality constraints and are the "bread and butter" of OR. Stochastic programming addresses issues dealing with uncertainty. Common approaches to solving IP and MIP entail solving several LP's, which are relaxations of the original IP or MIP that provide guidance and tighten bounds for branch and bound techniques. Similarly, Stochastic programming entails solving several LP's that represent different scenarios in the future — one can determine the best course of action in the present by optimizing the expected performance for the different scenarios.

The complexity of LP was not known for a long time. In the 70's, Klee and Minty created an example that showed that the simplex method can require exponential time. However, despite its worst-case exponential complexity, the simplex method generally performs very well in practice. In the late 70's, Khachian developed a polynomial-time algorithm for linear programming. On practical problems, however, this method was much less efficient than the simplex method. In 1984, Karmarkar devised an interior point method that is more efficient and can outperform the simplex method on certain large problems instances. Still, the simplex method is often the method of choice. During the announcement of the new release of CPLEX at the main OR conference, IN-

---

[1] Ironically, Dantzig was not considered for the Nobel Prize in Economics for work related to the discovery and application of LP. The prize was given to Koopmans and Kantorovich for their work applying LP to solve problems in economics.

3

FORMS, simplex based methods were shown to be very competitive or even outperforming interior point based methods on several benchmarks.[2]

Successful solutions of large-scale MIPs require formulations whose LP relaxations give a good approximation to feasible solutions. For instance, it is known that the Knapsack problem is relatively easy to solve if using the "right" LP formulations whose relaxations are very insightful for a branch and bound algorithm. However, some formulations of the Knapsack problem lead to poor relaxations of the corresponding LP, in the sense that they do not provide much information for a branch and bound algorithm.

The objective function is very important in OR models. In fact, in a recent review of Mathematical Programming, Dantzig (1991) emphasizes that, apart from LP and the simplex method, one of his main contributions was the formulation of an explicit *goal or objective function* to guide the search for feasible solution, instead of ad hoc ground rules. The objective function is essential in OR models, for two reasons: On one hand, it provides a criterion for optimization and it guides the search for solutions. Furthermore, it is a way of considering soft constraints. OR experts dealing with real-world applications use the approach of encoding constraints through the objective function, avoiding the use of hard constraints as much as possible. A goal constraint is an objective that is desirable but, if necessary, it can be violated. Goal Programming involves different techniques to produce solutions involving goal constraints. An example is the use of penalties associated with variables that measure the deviation between the desired goal and the actual value.

The work of Dantzig and Wolfe on solving LP by means of *Decomposition* has had a major impact on solving large-scale problems. In fact, even though the simplex method can handle sparse problems with several thousands of rows quite comfortably, it does not scale up when it comes to truly huge problems. For such problems the simplex method is out of the question and the Dantzig-Wolfe decomposition is needed. An example of the application of such decomposition methods is column generation techniques. They have been successfully applied, *e.g.* in Airline Crew Scheduling (see *e.g.*, Barnhart *et al.* 1994). Branch-and-price is an example of a column generation technique.

In a crew scheduling problem sequences of flights (pairings) are assigned to crews so that each flight for a specific fleet of airplanes is assigned to exactly one crew. Since pairings are subject to complicated rules (safety and contractual rules) it would be difficult to express constraints and costs if a direct encoding were used.[3] Instead, valid pairings are enumerated and the problem is formulated as a set partioning problem (SPP). In this formulation each column or variable corresponds to a pairing and the objective is to partition all of the flights into a set of minimum cost pairings.

The main drawback is that the number of pairings grows exponentially with

---

[2]CPLEX presentation of the new release at Informs, Montreal, 1998.

[3]By direct encoding we mean a formulation with variables x_{ij}, where $x_{ij} = 1$ if crew i is assigned to flight j.

the number of flights. For example, Vance (1993) found more than 5 million valid pairings in a daily problem with 253 flights. Problems with 1000 flights, a typical size for a U.S. domestic carrier, are likely to have billions of pairings.

The approach used to solve this formidable problem uses Dantzig-Wolfe column generation. The LP relaxation of the SPP is solved, but only a subset of columns are initially considered. This problem is called the *restricted master problem*. New columns are generated only as needed, and if needed, and based on the information provided by the solution to the *restricted master problem*, *i.e.*, the *dual prices*. These *dual prices* allow one to determine which flights should be included in "good" columns for the master problem. The problem of generating new columns is called the *subproblem* or *pricing problem*.

Duality

Duality plays an important role in OR. The theory of duality is elegantly developed in the context of LP. The basic idea is that every problem can be considered from a dual perspective — maximizing the profit is equivalent to minimizing costs. Every maximization LP problem gives rise to a minimization LP problem, its *dual*. Interestingly, every feasible solution of one problem provides a bound on the optimal value of the other problem, and if one of them has an optimal solution, so does the other and their optimal values are the same. This is what the famous Duality Theorem states, formally proved by Gale, *et al.* (1951). Its notions originated in conversations between Dantzig and von Neumann in the fall of 1947.

The theory of duality is also used to perform sensitivity analysis and parametric analysis, *i.e.*, the study of the impact on the objective function when the level of resources (the right hand sides of the linear constraints) vary or when the coefficients of the objective function vary. The technique of *penalties* uses sensitivity analysis to tighten bounds during branch-and-bound search.

Structure

Another theme in OR is to exploit inherent problem structure. *Trans-Shipment Problems* or *Network Flow Problems* are notable examples of the importance of exploiting structure. The special structure of these problems allows for very efficient (polynomial) algorithms. An interesting aspect of Network Flow Problems is that the optimal solution of instances involving only integral constraints are guaranteed to be also integer-valued. Many combinatorial problems, well beyond cases that deal with physical shipments of some commodity, such as scheduling problems, can be efficiently formulated as Network Flow Problems.

Typically, when using OR methods, one starts by categorizing the problem into a class of problems for which "good" solution methods have been developed

such as LP, Network Flow, or 0-1 programming problems. At a second level, in general using an automated process, structure is detected using inference methods. For example, when solving IP's or MIP's, the derivation of "cutting planes" is very important to eliminate parts of the search space that are guaranteed not to contain the optimal solution. Cutting planes are linear inequalities that can be added to the original formulation of an IP with the guarantee that no integer solution will be eliminated, but with the advantage of eliminating fractional solutions generated by the linear relaxation of the problem. The addition of cutting planes leads to tighter relaxations, and therefore their solutions are better approximations of the IP's solution. Gomory (1963) pioneered this approach, showing how to systematically generate "cuts" that lead to an integer solution.

Related to the approach of exploiting structure is the strategy of decomposing complex problems into simpler problems for which there are good algorithmic approaches or, at least, relaxed versions of the subproblems can be solved using efficient algorithms. Network Flow Problems play an important role in decomposition strategies since they represent a large class of problems after abstracting away some "details".[4]

Decomposition is often used to get tighter bounds for branch-and-bound methods. Such an approach is used, for example, by one of the fastest job-shop scheduling algorithms (Carlier and Pinson 1990). This algorithm bounds its search with Jackson's preemptive schedule algorithm for a single machine. Another example is the Knapsack problem. Even though this problem is NP-complete, it is relatively easy to solve in practice. It is used in several approaches, for example to solve the generalized machine assignment problem.

3 Opportunities for Integration of AI/OR

Solving large real-world scheduling problems has so far been almost exclusively the domain of operations research, but recent developments in constraint-satisfaction techniques have shown that they can be competitive on real-world problems. The constraint-satisfaction approach brings a novel perspective to planning and scheduling. Constraint-based methods provide a richer representational formalism compared to the traditional OR methods. Furthermore, constraint satisfaction techniques have developed powerful inference methods that lead to efficient variable domain reductions.

For example, the constraint programming language ILOG is now being used in actual fielded applications, in areas such as manpower and service scheduling, airline scheduling, cutting-stock in the steel industry, manufacturing scheduling

---

[4]Unfortunately, Network Flow Algorithms cannot be used when there are global constraints on the nodes of the network. An example of a global constraint would state that the amount of goods shipped through certain nodes corresponds to 30 % of the total amount of goods shipped.

for the auto industry, supply chain management, etc. Companies such as SAP, Peoplesoft, and I2, leading developers of software solutions for managing human resources, accounting, materials management, distribution, and manufacturing, across different industries, combine different optimization techniques such as constraint programming, mathematical programming, and local search methods. These new developments have created a unique opportunity to investigate the integration of AI, primarily constraint-satisfaction methods, and OR techniques. Some key issues are outlined in the following paragraphs.

Hybrid Solvers

This is an important emerging area of research combining CSP techniques with OR techniques. Work in this area began with CLP(R), Prolog III, and Chip, combining constraint satisfaction (CSP) methods with linear programming. The ILOG system integrates a finite domain propagation solver for discrete variables with CPLEX, for continuous variables. Promising results have been obtained using such *hybrid* approaches, which allow for more powerful constraint reasoning: consistency checking and domain reduction techniques enforce efficient constraint propagation, while linear programming relaxations provide infeasibility, or bounds on the objective function. For example, a research team at Imperial College reports that *only* by using a hybrid approach were they able to solve to optimality hoist scheduling problems (Rodosek and Wallace 1998). These problems could not be solve optimally in the OR literature. McAloon *et al.* (1998) also report similar benefits of using hybrid solvers to solve a multi-commodity integer network flow problem of the Dutch Railways which is greatly complicated by additional constraints on the coupling and decoupling of trains.

Duality

The notion of duality expresses the fact that there are two complementary ways of looking at a problem. Duality is a powerful concept that has been extensively exploited by the OR community in linear programming. Duality can be exploited to solve problems, by considering simultaneously two perspectives — the primal and dual view of the problem. Such approaches, in general, allow for stronger inferences, both in terms of cutting planes as well as variable domain reductions. Recently there have been several promising results in the CSP community using a dual formulation approach (*e.g.*, to solve hard timetabling problems, McAloon *et al.* 1997, Gomes *et al.* 1998). However, in general, duality is not yet well understood for problems involving constraints other than inequality constraints. Research in this area, coupled with the study of the design of global constraints and good relaxation schemes for primal and dual formulations, is very promising. The study of new ways for performing sensitivity analysis based on duality is also a promising research area.

7

Problem Structure

In general structured models are easier to understand and compute with. The OR community has identified several classes of problems with very a interesting, *tractable*, structure. LP, and Network Flow problems are good examples of such problems. OR also exploits the structure of problems during inference by generating, in generally automatically, "cutting planes", which allow for tighter relaxations that are therefore closer to the optimal integer solution. The CSP community, on the other hand, has identified the special structure of several global constraints that are ubiquitous in several problems, which allow the development of efficient constraint propagation techniques for the reduction of the variable domains.

In general, however, the notion of structure is very hard to define, even though we recognize structure when we see it. For example, there is not a methodology that shows how to construct good cutting planes. Formalizing the notion of structure and understanding its impact in terms of search is a big challenge for both AI and OR. AI has made some progresses in this area, namely in the study of phase transition phenomena, correlating structural features of problems. For example, in the Satisfiability problem it is known that the difficulty of problems depends on the ratio between number of clauses and number of variables (Kirkpatrick and Selman 1994).

In Gomes and Selman (1997) we introduce the Quasigroup Completion problem (QCP), a structured framework for studying search methods. Our study of QCP reveals that its complexity depends critically on the percentage of pre-assigned cells to the initial matrix of the quasigroup. Depending on the level of pre-assignment, we can identify different levels of difficulty for the instances of the QCP problem, namely an under-contrained area (low levels of pre-assignment) and an over-constrained area (high levels of pre-assignment) where problem instances are relatively easy to solve. The critically constrained area corresponds to intermediate levels of pre-assignment of colors to the initial quasigroup matrix and those instances tend to be relativley harder to solve. (See also Appendix 1 for a description of QCP).

Recently, Monasson *et al.* (1999) also showed that random satisfiability instances that are a mixture of 2-Sat and 3-Sat clauses, the 3-Sat clauses with weight p, scale linearly as long as $p <= 0.4$. Another structural feature that Monasson *et al.* recently formalized is the concept of *backbone*. The backbone of an instance corresponds to the shared structure of all the solutions of a problem instance. In other words, the set of variables and corresponding assignments that are common in all the solutions of a problem instance is the backbone.

As a final remark it is important to mention the trade-off between highly structured models, which tend to be very specific and therefore fit a narrow class of problems, and more unstructured models that are more flexible and therefore easier to fit real world problems.

8

Local Search

Local search methods or meta-heuristics are often used to solve challenging combinatorial problems. Such methods start with an initial solution, not necessarily feasible, and improve upon it by performing small "local" changes. One of the earliest applications of local search was to find good solutions for the Traveling Salesman Problem (TSP) (Lin (1965) and Lin and Kernighan (1973)). Lin and Kernighan showed that by performing successive swaps of cities to an arbitrary initial tour of cities, until no such swaps are possible, one can generate solutions that are surprisingly close to the shortest possible tour. There are several ways of implementing local search methods, depending on the choice of the initial solution, types of "local" changes allowed, and feasibility and cost of (intermediate) solutions.

There is a great deal of overlap in research on local search by the AI and OR communities, namely in simulated annealing (Kirkpatrick *et al.* 1983), tabu search (Glover 1989), and genetic algorithms (Holland 1975). A recent new area of application for local search methods is in solving NP-complete *decision problems*, such as the Boolean satisfiability (SAT) problem. In 1992, Selman *et al.* showed that a greedy local search method, called GSAT, could solve instances with up to 700 hundred variables. Currently GSAT and variants (*e.g.*, WALKSAT) are among the best methods for SAT, enabling us to solve instances with up to 3000 variables (Selman *et al.* 1994). Closely related work in the area of scheduling is the technique of "MinConflicts" proposed by Minton *et al.* (1992).

Local search, and mixtures of local and global search strategies have proved to be very effective to tackle real world problems, in general beyond the reach of pure complete search methods.

Randomization

Stochastic strategies have been very successful in the area of local search. However, local search procedures are inherently incomplete methods. An emerging area of research is the study of Las Vegas algorithms, *i.e.*, randomized algorithms that always return a model satisfying the constraints of the search problem or prove that no such model exists (Motwani and Raghavan 1995). The running time of a Las Vegas style algorithm can vary dramatically on the same problem instance. The extreme variance or "unpredictability" in the running time of complete search procedures can often be explained by the phenomenon of "heavy-tailed cost distributions". The understanding of these characteristics explains why "rapid restarts" and portfolio strategies are very effective. Restart and protfolio strategies eliminate the heavy-tailed behavior and exploit *any significant probability mass early on in the distribution*. Restarts and portfolio strategies therefore reduce the variance in runtime and the probability of failure

of the search procedures, resulting in more robust overall search methods (Frost et al. 1997; Gomes and Selman 1999; Gomes et al. 1998; Gomes et al. 2000; and Hoos 1999).

In the first paper of the Appendix, "Heavy-tailed Distributions in Combinatorial Search", we show that backtrack style algorithms are often characterized by distributions that can have infinite moments called heavy-tailed distributions.

In the second paper of the Appendix, "Boosting Combinatorial Search Through Randomization" we discuss how to exploit heavy-tailed behavior to speed up search by using restart strategies.

In the third paper of the Appendix, "Algorithm Portfolios", we discuss portfolio strategies, in the context of Mixed Integer Programming.

Cutting planes and constraint propagation

OR's inference method of choice, during search, is "cuts". "Cuts" or "cutting planes" are *redundant* constraints, in the sense that they do not eliminate feasible solutions. However, although these constraints are redundant in terms of the solution, they can play a major role during the search process. A classical example of the importance of cutting planes involves the pigeonhole problem: by adding the appropriate redundant constraints to a linear programming formulation, its relaxation immediately returns infeasibility. Without such redundant constraints, the results of the LP relaxation are useless. Gomory (1963) pioneered the study of cutting planes, showing how to systematically generate "cuts" that lead to an integer solution. The OR community has developed several techniques for the generation of cuts, but, in general, it is not clear how to construct such cuts.

The CSP's community, on the other hand, mainly relies on domain reduction techniques for inference during search. A very successful strategy is to exploit the structure of special constraints and treat them as a global constraint (Beldiceanu and Contejean 1994, Caseau and Laburthe 1997; Regin 1994 and 1996). Some examples of such propagation methods are the constraint that guarantees that all elements of a vector are different (all-different constraint) and the constraint that enforces that certain values occur a given number of times in a given vector of variables (cardinality constraint). The implementation of such constraints is an interesting use of Network Flow algorithms (Regin 1994, 1996).

A direction of research is the study of techniques that will lead to the generation of better cuts as well as efficient domain reduction techniques, and the combination of cuts with domain reduction techniques. Relevant work in this area is that of Lovasz and Schrijver (1991) and Balas, Ceria, and Cornuejols (1993). They have developed the *lift-and-project* technique. Hooker (1992) has developed cutting plane algorithms for IP and resolution methods in propositional logic. Work on the automated generation of cutting planes for problems

such as the pigeonhole problem has been done by Barth (1996). The work done at Kestrel Institute using a transformational approach to scheduling encompasses the generation of very efficient constraint propagation techniques (Smith and Parra 93). Relevant work on exploiting the structure of global constraints for domain reduction is that of *e.g.*, Beldiceanu and Contejean 1994, Caseau and Laburthe 1997, and Regin (1994,1996).

Coupling of Column Generation with CSP

As described above, the column generation formulation involves two phases: (1) generating columns, and (2) solving the corresponding LP relaxation problem. In general, the process of generating columns is quite "messy", since complicated constraints are involved. OR methods are not suitable for such a task. A combination of AI and OR techniques can enhance this phase considerably. Leconte *et al.* (1997) have reported very good results for solving a column generation problem applied to bin-packing configuration problems using hybrid solvers.

Robustness

Ideally, we would like to find not only *good* but also *robust* solutions. The intuition behind robustness is: given a set C of changes to the initial formulation of the problem instance, a solution A is more robust than solution B, w.r.t. set C, if the number of changes required to fix solution A is less than the number of changes required to fix solution B. There are very few results on the study of robustness. Most results emphasize generation of solutions from scratch completely ignoring issues on robustness. This is an area that requires substantial research, starting with a good definition of the notion of robustness.

4 Synopsis of the papers in the Appendix

In the Appendix we include three papers that elaborate on several issues discussed above. In this section we give a short synopsis of each paper.

Heavy-tailed Distributions in Combinatorial Search (with Bart Selman and Nuno Crato) In this paper we show that backtrack style algorithms are often characterized by distributions that have infinite moments called heavy-tailed distributions. We also show how restart strategies are effective for eliminating heavy-tailed behavior. In fact, the heavy-tailed behavior that characterizes combinatorial search methods can be exploited in terms of algorithm design, leading to considerable speed ups in runtime. This issue is expanded in the next two papers of the Appendix.

Boosting Combinatorial Search through Randomization (with Henry Kautz and Bart Selman) In this paper we discuss how restart strategies are effective to speed up search, taking advantage of heavy-tailed behavior. We consider applications in planning and scheduling.

Algorithm Portfolios (with Bart Selman) In this paper we discuss portfolio strategies, in the context of Mixed Integer Programming.

References

Alt, H., Guibas, L., Mehlhorn, K., Karp, R., and Wigderson A. (1996) A method for obtaining randomized algorithms with small tail probabilities. *Algorithmica*, 16, 1996, 543–547.

Barnhart, C., Johnson, E., Nemhauser, G., Savelsbergh, M., and Vance, P. (1994) Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Mathematical Programming. State of the Art* Birge, J., and Murty, K. (eds.), 1994, 186-207.

Balas, E., Ceria, S., and Cornuejols, G. (1993) A lift and project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming* 58 (1993), 295-324.

Barth,P. (1996) Logic based 0-1 Constraint programming. Kluwer, 1996.

Beldiceanu N. and Contejean, E. (1994) Introducing global constraints in CHIP. *Mathl. Comput. Modelling*, 20 (12), 1994, 97–123.

Carlier,J. and Pinson, E. (1990) A practical use of Jackson's preemptive schedule for solving the job shop problem. *Annals of operations Research*, 26, 1990, 269-287.

Caseau, Y. and Laburthe, F. (1997) Solving various weighted matching problems with constraints. *Principles and Practice of Constraint Programming*, vol. 1330 of *Lecture Notes in Computer Science*, 1997, 17–31.

Caseau, Y., Laburthe, F., Le Pape, C., and Rottembourg B. (2000) Combining local and global search in a constraint programming environment. *Knowledge Engineering Review*, Vol. 15 (1), 2000.

Clements D., Crawford J., Joslin D., Nemhauser G., Puttlitz, M, and Savelsbergh, M. (1997) Heuristic Optimization: a Hybrid AI/OR Approach. *Workshop of Constraint Programming*, Austria, 1997.

Dantzig, G.B. (1991) Linear Programming. *History of Mathematical Programming, A collection of Reminiscences* Lenstra, J., Kan, R., Schrijver, A. (eds.), CWI, Amsterdam, 1991, 19-31.

Dantzig, G.B. and Wolfe, P. (1960) Decomposition principle for linear programs. *Operations Research*, 8, 1960, 101-111.

Dixon, H. and Ginsberg, M. (2000) Combining satisfiability techniques from AI and OR. *Knowledge Engineering Review*, Vol. 15 (1), 2000.

Frost, D., Rish, I., and Vila, L. (1997) Summarizing CSP hardness with continuous probability distributions. *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 1997.

Gomes, C.P., Kautz, H., and Selman, B. (1998) Boosting Combinatorial Search Through Randomization. *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

Gomes, C.P. and Selman, B. (1997) Problem structure in the presence of perturbations. *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, New Providence, RI, 1997, 221–226.

Gomes, C.P. and Selman, B. (1999) Search Strategies for Hybrid Search Spaces. *Proceedings of the Eleventh International Conference on Tools with Artificial Intelligence (ICTAI-99)*, 1999.

Gomes, C.P., Selman, B., Crato, N, and Kautz, H. (2000) Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems. To appear in *Journal of Automated Reasoning* 2000.

Gomes, C.P. and Selman, B., McAloon, K., and Tretkoff C. (1998b). Randomization in Backtrack Search: Exploiting Heavy-Tailed Profiles for Solving Hard Scheduling Problems. *Proc. AIPS-98*.

Gomory, R. (1958) An outline of algorithm for integer solutions to linear programs. *Bulletin of the American mathematical Society*, 1958, 64, 275-278.

Gomory, R. (1963) An algorithm for integer solutions to linear programs. *Recent Advances in Mathematical Programming* McGraw-Hill, Graves, R. and Wolfe, P. (eds.) 1963, 64, 260-302.

Hoos, H. (1999) On the Run-time Behaviour of Stochastic Local Search Algorithms for SAT. *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-99)*, 1999, 661-666.

Hooker, J. (1992) Generalized resolution for 0-1 linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 6, 1992, 271-286.

Hooker, J., Ottosson, G., Thorsteinsson, E. and Kim, H. (2000) A scheme for unifying optimization and constraint satisfaction methods. *Knowledge Engineering Review*, Vol. 15 (1), 2000.

Jeroslow, R. (1980) A cutting plane game for facial disjunctive programs. *SIAM J. Control and Optimization*, 18, 1980, 264-280.

Kautz, H. and Walser, J. (2000) Integer Optimization Models of AI Planning Problems. *Knowledge Engineering Review*, Vol. 15 (1), 2000.

Lovasz, L. and Schrijver A. (1991) Cones of matrices and set functions and 0-1 optimizations. *SIAM J. Control and Optimization*, 1991, 166-190.

Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., and Troyansky, L. (1996). Determining computational complexity from characteristic 'phase transitions'. *Nature*, Vol. 400(8), 1999.

Kantorovich, V. (1939) Mathematical Methods in the organization of planning of production. *Management Science*, 6, 1960, 366-422 [English translation.]

Karmarkar, N. (1984) A new polynomial time algorithm for linear programming. *Combinatorica*, 4, 1984, 373-395.

Khachian, V. (1979) A polynomial time algorithm for linear programming. *Math. Doklady*, 20, 1979, 191-194. [English translation.]

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. (1983) Optimization by simulated annealing. *Science*, 220 (1983) 671–680.

Kirkpatrick, S. and Selman, B. (1994) Critical Behavior in the Satisfiability of Random Boolean Expressions. *Science*, 264 (May 1994) 1297–1301.

Klee, V. and Minty, G. (1972) How good is the simplex algorithm? *Inequalities-III* Shisha, O. (ed.) New York: Academic Press, 1972, 159-175.

Lin, S. (1965) Computer solutions of the traveling salesman problem. *BSTJ*, 44, no 10 (1965), 2245–69.

Lin, S. and Kernighan, B.W. (1973) An effective heuristic for the traveling-salesman problem. *Oper. Res.* 21 (1973) 498–516.

Luby, M., Sinclair A., and Zuckerman, D. (1993). Optimal speedup of Las Vegas algorithms. *Information Process. Lett.*, 17, 1993, 173–180.

McAloon, K., Tretkoff C. and Wetzel G. (1997). Sports League Scheduling. *Proceedings of Third Ilog International Users Meeting*, 1997.

McAloon, K., Tretkoff C. and Wetzel G. (1998). Disjunctive programming and and cooperating solvers. *Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search*, 1998. Kluwer, Woodruff, D. (ed.), 75-96.

Minton, S., Johnston, M., Philips, A.B., and Laird, P. (1992) Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58 (1992) 161–205,

Nemhauser, G., and Wolsey L. (1988) Integer and Combinatorial Optimization. John Wiley, New York (1988).

Nemhauser, G., and Trick, M. (1997) Scheduling a major college basketball conference. Georgia Tech., Technical Report, 1997.

Oddi A. and Smith, S. (1997) Stochastic procedures for generating feasible schedules. *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, New Providence, RI, 1997.

Puget, J-F., and Leconte, M. (1995). Beyond the Black Box: Constraints as objects. *Proceedings of ILPS'95*, MIT Press, 513–527.

Regin J.C. (1994). A filtering algorithm for constraints of difference in CSPs. *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-94)*, Seattle, 1994..

Regin J.C. (1996). Generalized arc consistency for global cardinality constraint. *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, Oregon, 1996.

Rodosek, R., and Wallace, Mark (1998). One Model and different solvers for hoist scheduling problems. Manuscript in preparation.

Smith, D., Frank, J., and Jónsson A. (2000) Bridging the gap Between Planning and Scheduling. *Knowledge Engineering Review*, Vol. 15 (1), 2000.

Smith D., and Parra E. (1993). Transformational Approach To Transportation Scheduling. *Proceedings of the Eigth Knowledge-Based Software Engineering Conference*, 1993, Chicago,

Vance, P., (1993) Crew Scheduling, Cutting Stock, and Column Generation: Solving Huge Integer Programs. Georgia Tech., PhD Thesis, 1993.

Vossen, T., Ball, M., Lotem, A., and Nau D. (2000) Applying Integer Programming Techniques to AI Planning. *Knowledge Engineering Review*, Vol. 15 (1), 2000.

Appendix

Heavy-Tailed Distributions in Combinatorial Search

Carla P. Gomes[1], Bart Selman[2], and Nuno Crato[3]

[1] Computer Science Department, Cornell University, Ithaca, NY 14853,
gomes@cs.cornell.edu
[2] Computer Science Department, Cornell University, Ithaca, NY 14853,
selman@cs.cornell.edu
[3] Dept. of Mathematics, New Jersey Institute of Technology, Newark, NJ 07102,
USA, ncrato@m.njit.edu

Abstract. Combinatorial search methods often exhibit a large variability in performance. We study the cost profiles of combinatorial search procedures. Our study reveals some intriguing properties of such cost profiles. The distributions are often characterized by very long tails or "heavy tails". We will show that these distributions are best characterized by a general class of distributions that have no moments (i.e., an infinite mean, variance, etc.). Such non-standard distributions have recently been observed in areas as diverse as economics, statistical physics, and geophysics. They are closely related to fractal phenomena, whose study was introduced by Mandelbrot. We believe this is the first finding of these distributions in a purely computational setting. We also show how random restarts can effectively eliminate heavy-tailed behavior, thereby dramatically improving the overall performance of a search procedure.

1 Introduction

Combinatorial search methods exhibit a remarkable variability in the time required to solve any particular problem instance. For example, we see significant differences on runs of different heuristics, runs on different problem instances, and, for stochastic methods, runs with different random seeds. The inherent exponential nature of the search process appears to magnify the unpredictability of search procedures. It is not uncommon to observe a combinatorial method "hang" on a given instance, whereas a different heuristic, or even just another stochastic run, solves the instance quickly.

---
[0]

We explore the cost distribution profiles of search methods on a variety of problem instances. Our study reveals some intriguing properties of such cost profiles. The distributions are often characterized by very long tails or "heavy tails". We will show that these distributions are best captured by a general class of distributions that have no moments, *i.e.*, they have infinite mean, variance, etc.

Heavy-tailed distributions were first introduced by the Italian-born Swiss economist Vilfredo Pareto in the context of income distribution. They were extensively studied mathematically by Paul Lévy in the period between the world wars. Lévy worked on a class of random variables with heavy tails of this type, which he called *stable* random variables. However, at the time, these distributions were largely considered probabilistic curiosities or pathological cases mainly used in counter-examples. This situation changed dramatically with Mandelbrot's work on fractals. In particular, two seminal papers of Mandelbrot (1960, 1963) were instrumental in establishing the use of stable distributions for modeling real-world phenomena.

Recently, heavy-tailed distributions have been used to model phenomena in areas as diverse as economics, statistical physics, and geophysics. More concretely, they have been applied in stock market analysis, Brownian motion, wheather forecasts, earthquake prediction, and recently, for modeling time delays on the World Wide Web (e.g., Mandelbrot 1983; Samorodnitsky and Taqqu 1994). We believe our work provides the first demonstration of the suitability of heavy-tailed distributions in modeling the computational cost of combinatorial search methods.
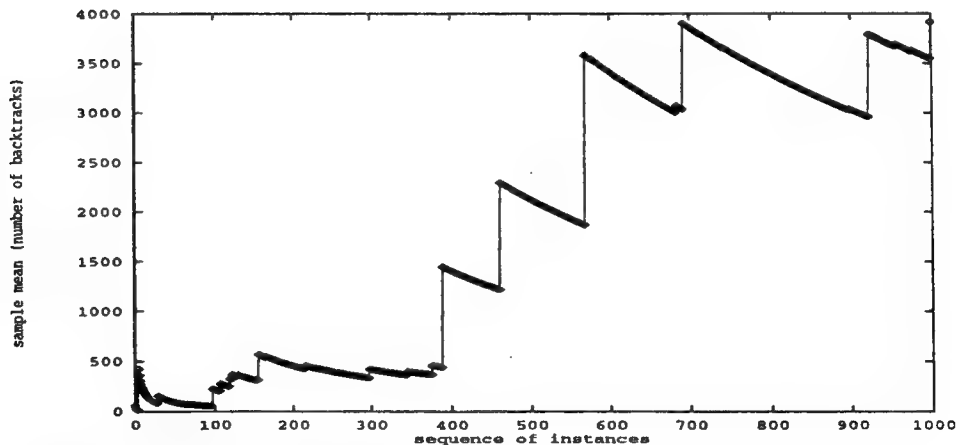

Figure 1a: Erratic behavior of mean cost value.

Various researchers studying the computational nature of search problems have informally observed the erratic behavior of the mean and the variance
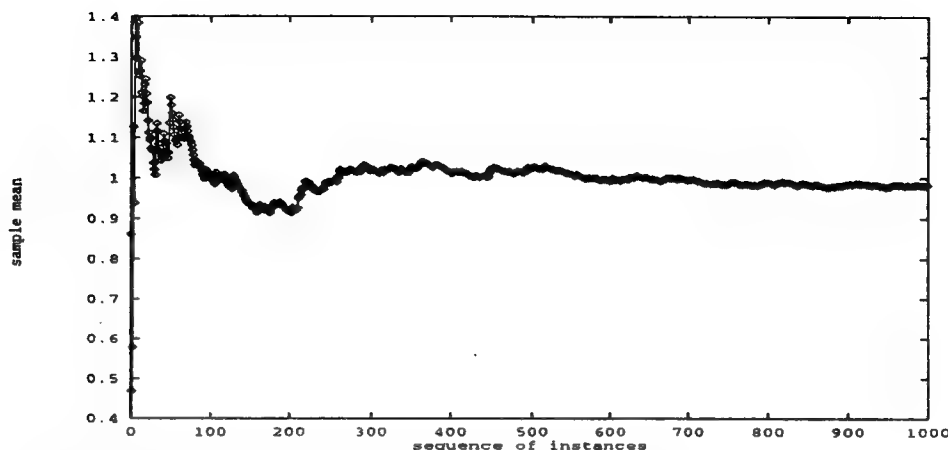
Figure 1b: Mean for a standard distribution (gamma).

of the search cost. This phenomenon has led them to use the median cost to characterize search difficulty. The heavy-tailed distributions provide a formal framework explaining the erratic mean and variance behavior. See Figure 1, for a preview of this phenomenon. Figure 1a shows the mean cost calculated over an increasing number of runs, on the same instance, of a backtrack style search procedure (details below). Contrast this behavior with that of the mean of a standard probability distribution (a gamma distribution; no heavy tails) as given in Figure 1b. In Figure 1b, we see that the sample mean converges rapidly to a constant value with increasing sample size. On the other hand, the heavy-tailed distribution in Figure 1a shows a highly erratic behavior of the mean that does not stabilize with increasing sample size.[1]

As a direct practical consequence of the heavy-tailed behavior of cost distributions, we show how randomized *restarts* of search procedures can dramatically reduce the variance in the search behavior. In fact, we will demonstrate that a search strategy with restarts can eliminate heavy-tailed distributions. This may explain the common informal use of restarts on combinatorial search problems.

2   Structured Search Problems

The study of the complexity and performance of search procedures when applied to realistic problems is greatly hampered by the difficulty in gathering realistic data. As an alternative, researchers heavily resort to randomly generated instances or highly structured problems from, *e.g.*, finite algebra. The random instances clearly lack sufficient structure, whereas the finite algebra problems are, in some sense, too regular. In order to bridge this gap, we introduced a new

---

[1] The median, not shown here, stabilizes rather quickly at the value 1.

benchmark domain, the *Quasigroup Completion Problem* (Gomes and Selman 1997a).

A quasigroup is an ordered pair $(Q, \cdot)$, where Q is a set and $(\cdot)$ is a binary operation on Q such that the equations $a \cdot x = b$ and $y \cdot a = b$ are uniquely solvable for every pair of elements a, b in Q. The *order* N of the quasigroup is the cardinality of the set Q. The best way to understand the structure of a quasigroup is to consider the N by N multiplication table as defined by its binary operation. The constraints on a quasigroup are such that its multiplication table defines a *Latin square*. This means that in each row of the table, each element of the set Q occurs exactly once; similarly, in each column, each element occurs exactly once (Denes and Keedwell 1974).

An *incomplete* or *partial latin square* P is a partially filled N by N table such that no symbol occurs twice in a row or a column. The Quasigroup Completion Problem is the problem of determining whether the remaining entries of the table can be filled in such a way that we obtain a complete latin square, that is, a full multiplication table of a quasigroup. We view the pre-assigned values of the latin square as a *perturbation* to the original problem of finding an arbitrary latin square. Another way to look at these pre-assigned values is as a set of additional problem constraints on the basic structure of the quasigroup.

There is a natural formulation of the problem as a Constraint Satisfaction Problem. We have a variable for each of the N^2 entries in the multiplication table of the quasigroup, and we use constraints to capture the requirement of having no repeated values in any row or column. All variables have the same domain, namely the set of elements Q of the quasigroup. Pre-assigned values are captured by fixing the value of some of the variables.

Colbourn (1983) showed the quasigroup completion problem to be NP-complete. In previous work, we identified a clear phase transition phenomenon for the quasigroup completion problem (Gomes and Selman 1997a). See Figures 2 and 3. From the figures, we observe that the costs peak roughly around the same ratio (approximately 42% pre-assignment) for different values of N. (Each data point is generated using 1,000 problem instances. The pre-assigned values were randomly generated.) This phase transition with the corresponding cost profile allows us to tune the difficulty of our problem class by varying the percentage of pre-assigned values.

An interesting application area of latin squares is the design of statistical experiments. The purpose of latin squares is to eliminate the effect of certain systematic dependency among the data (Denes and Keedwell 1974). Another interesting application is in scheduling and timetabling. For example, latin squares are useful in determining intricate schedules involving pairwise meetings among the members of a group (Anderson 1985). The natural perturbation of this problem is the problem of completing a schedule given a set pre-assigned meetings.

The quasigroup domain has also been extensively used in the area of automated theorem proving. In this community, the main interest in this domain has been driven by questions regarding the existence and nonexistence of quasi-

groups with additional mathematical properties (Fujita et al. 1993; Lam et al. 1989).
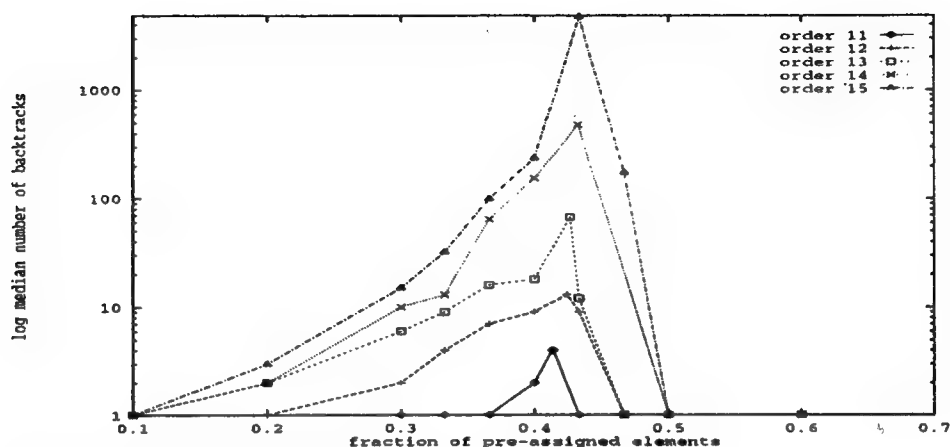

Figure 2: The Complexity of Quasigroup Completion (Log Scale).

3   Computational Cost Profiles

In this section, we consider the variability in search cost due to different search heuristics. As our basic search procedure, we use a complete backtrack-style search method. The performance of such procedures can vary dramatically depending on the way one selects the next variable to branch on (the "variable selection strategy") and in what order the possible values are assigned to a variable (the "value selection strategy").

One of the most effective strategies is the so-called First-Fail heuristic.[2] In the First-Fail heuristic, the next variable to branch on is the one with the smallest remaining domain (*i.e.*, in choosing a value for the variable during the backtrack search, the search procedure has the fewest possible options left to explore — leading to the smallest branching factor). We consider a popular extension of the First-Fail heuristic, called the Brelaz heuristics (Brelaz 1979), which was originally introduced for graph coloring procedures.

The Brelaz heuristic specifies a way for breaking ties in the First-fail rule: If two variables have equally small remaining domains, the Brelaz heuristic proposes to select the variable that shares constraints with the largest number of the remaining unassigned variables. A natural variation on this tie-breaking rule

---

[2] This is really a prerequisit for any reasonable backtrack-style search method. In theorem proving and Boolean satisfiability, the rule is related to the powerful unit-propagation heuristic.
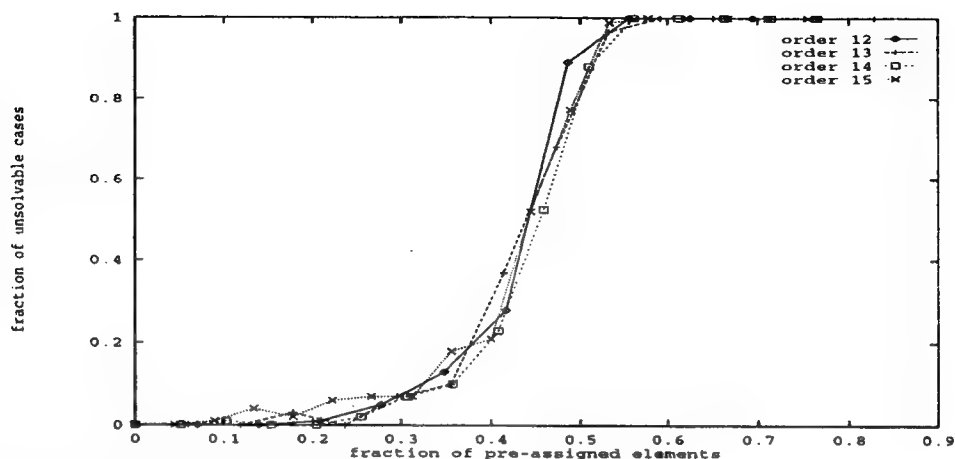
21

Figure 3: Phase Transition for the Completion Problem.

is what we call the "reverse Brelaz" heuristic, in which preference is given to the variable that shares constraints with the *smallest* number of unassigned variables. Any remaining ties after the (reverse) Brelaz rule are resolved randomly. (Note that such tie breaking introduces a stochastic element in our complete search method.) One final issue left to specify in our search procedure is the order in which the values are assigned to a variable. In the standard Brelaz, value assignment is done in lexicographical order (*i.e.*, systematic). In our experiments, we consider four strategies:

- *Brelaz-S* — Brelaz with systematic value selection,
- *Brelaz-R* — Brelaz with random value selection,
- *R-Brelaz-S* — Reverse Brelaz with systematic value selection, and
- *R-berlaz-R* — Reverse Brelaz with random value selection.

We encoded this problem in C++ using ILOG SOLVER, a powerful C++ constraint programming library (Puget 1994). ILOG provides a backtracking mechanism that allows us to keep track of variables and their domains, while maintaining arc-consistency (van Hentenryck *et al.* 1992).

Figure 4 shows the performance profile of our four strategies for an instance of the quasigroup completion problem of order 20 with 10% pre-assigned values, *i.e.*, in the underconstrained area. Each curve gives the cumulative distribution obtained for each strategy by solving the problem 10,000 times. The cost (horizontal axis) is measured in number of backtracks, which is directly proportional to the total runtime of our strategies. For example, the figure shows that R-Brelaz-R, finished roughly 80% of the 10,000 runs in 15 backtracks or less.

First, we note that the (cumulative) distributions have surprising long tails after a steep initial climb. We will return to this issue below. We also see that that R-Brelaz-R dominates the other strategies over almost the full range of the distribution. (Brelaz-S dominates very early on but the difference is not
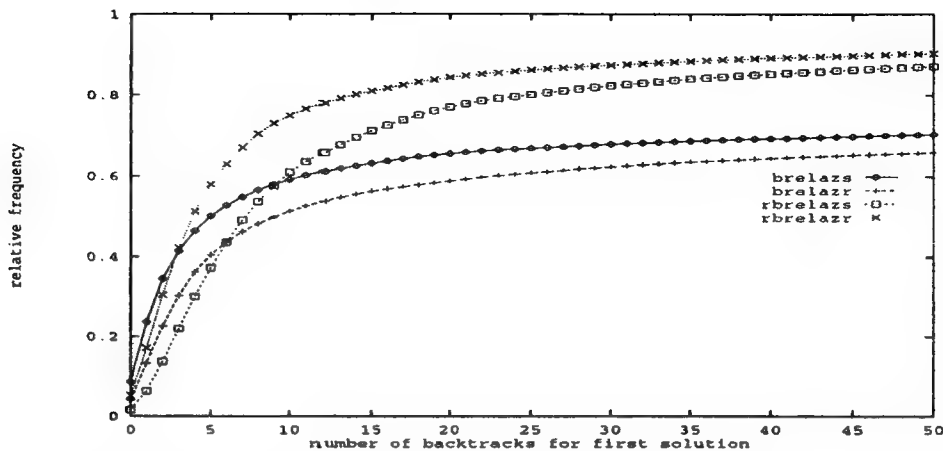
Figure 4: Finding quasigroups of order 20 with 10% pre-assigned values.

statistically significant.) Figure 5 shows the performance profile on an instance of the quasigroup completion problem in the critically constrained area. The initial climb followed by a long tail is even more dramatic. In this case, R-Brelaz-R and R-Brelaz-S give virtually the same performance, and both dominate the other two strategies.

These profiles suggest that it is difficult to take advantage of combining different heuristics in order to reduce variability. It was our initial intention to build so-called algorithm portfolios to reduce variability (Huberman *et al.* 1997 and Gomes and Selman 1997b). However, with one strategy dominating over the full profile there is no clear payoff in combining different heuristics, at least in this domain. In fact, it may well be the case that on a given problem domain, one can often find a single dominating heuristic. Our study here is not meant to be exhaustive regarding the full spectrum of search heuristics. In particular, we restricted ourselves to variations on the well-known Brelaz search heuristic.

In the next section, we concentrate on a perhaps more striking feature of the cost distributions: the *long tails*. As we will see in our section on "restarts", the heavy tail behavior can be exploited effectively to reduce variability in the search cost.

4   Heavy-Tailed Distributions

Figure 6a shows the heavy-tailed nature of our cost distributions in a more direct manner. The probability distribution was obtained using R-Brelaz-R on an instance of the quasigroup completion problem of order 20 with 5% preplacement.[3] We considered the distributions of over two dozen randomly picked instances

---

[3] Work on exceptionally hard problems provides further support for the heavy tailed nature of the distributions (Gent and Walsh 1993; Smith and Grant 1995). However,
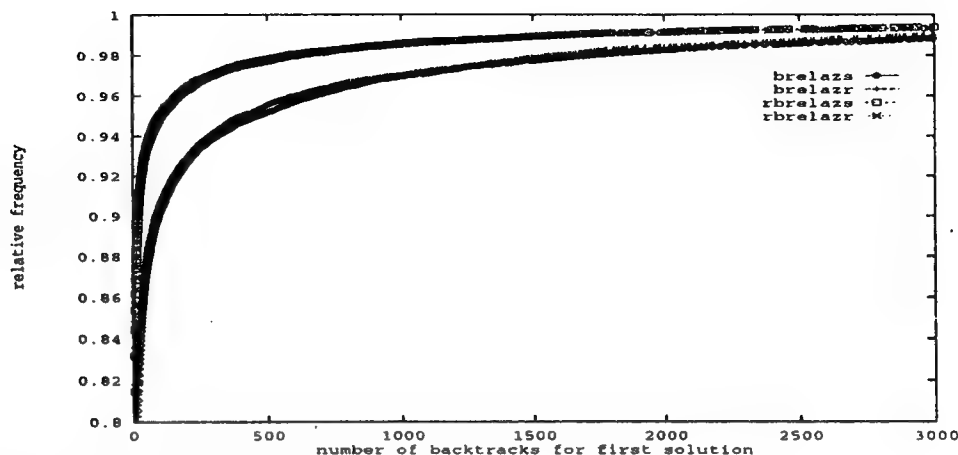
Figure 5: Finding quasigroups of order 10 at the phase transition.

from both the under-constrained and the critically constrained area, as well as some aggregate distributions. We found heavy-tailed distributions for almost all of our solvable instances and aggregate distributions. Some very easy solvable instances did not exhibit heavy tails. Interestingly, the unsolvable instances do not exhibit heavy-tails. The gamma and normal distributions were the best fit for the majority of our unsolvable instances (see also Frost et al. (1997)).

In order to model the long tail behavior of our distributions, we will consider distributions which asymptotically have tails of the Pareto-Lévy form, *viz.*

$$\Pr\{X > x\} \sim C.x^{-\alpha}, \quad x > 0 \tag{1}$$

where $\alpha > 0$ is a constant. These are distributions whose tails have a *hyperbolic decay*. For the case which concerns us it suffices to consider this tail behavior for the positive values of the random variable X. So, in what follows we will assume that the distribution has support on the positive half line, i.e., $\Pr\{0 \leq X < \infty\} = 1$.

Mandelbrot (1983) provides an excellent introduction to these distributions with a discussion of their inherently self-similar or fractal nature. For a complete treatment of stable distributions see either Zolotarev (1986), or the more modern approach of Samorodnitsky and Taqqu (1994). See also de Lima (1997). In what follows, we simply outline the main results we will need to use.

A random variable X is said to have a *stable distribution* if for any $n > 1$ there is a positive number C_n and a real number D_n such that

$$X_1 + X_2 + \cdots + X_n \overset{\mathcal{D}}{=} C_n X + D_n, \tag{2}$$

the heavy tails we observed appear more ubiquitous: We observed heavy-tails in the majority of solvable instances in the under-constrained area and also in the majority of solvable instances in the critically constrained area. For other recent related work on cost distributions, see Frost et al. (1997) and Kwan (1995).
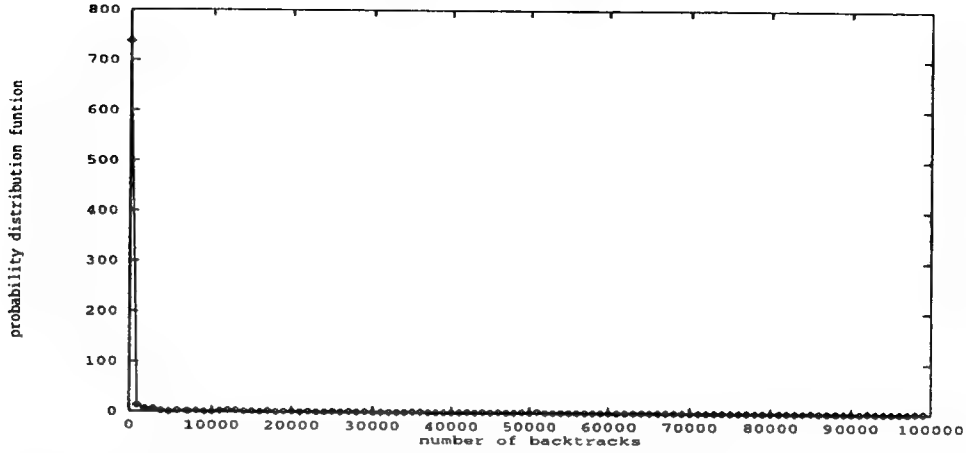
24

Figure 6a: Probability distribution exhibiting heavy-tailed behavior.

where $X_1, X_2, \ldots, X_n$ are independent copies of X and $\stackrel{\mathcal{D}}{=}$ stands for equality in distribution. From this definition, it can be shown that the following is implied

$$C_n = n^{1/\alpha} \tag{3}$$

for some $0 < \alpha \leq 2$ (Samorodnitsky and Taqqu 1994). The constant α is called the *index of stability* of the distribution. Stable distributions with $\alpha < 2$ have heavy tails of the Pareto-Lévy type. The index of stability is the same α which appears in equation (1).

Since the existence or nonexistence of moments is completely determined by the tail behavior, it is simple to check that the index of stability α is the *maximal moment exponent* of the distribution. For $\alpha < 2$, moments of X of order less than α are finite while all higher order moments are infinite, i.e., $\alpha = \sup\{a > 0 : \mathrm{E}|X|^a < \infty\}$. For example, when $\alpha = 1.5$, the distribution only has a finite mean but no finite variance. When $\alpha = 0.6$, the distribution does not have a finite mean nor a finite variance.

While it is relatively easy to define a stable distribution, only in a few particular cases the density of the stable distributions is known in its *closed* form.

It should be noted, however, that distributions with tails of the form (1) are in the *domain of attraction* of stable distributions, *i.e.*, properly normalized sums of variables with tails of the Pareto-Lévy type converge in distribution to an α-stable random variable. This additive character of stable distributions matches the additive nature of the number of nodes searched in subtrees of the backtrack tree. This provides some intuition behind the suitability of the stable distributions for modeling search cost distributions.

In order to check for the existence of heavy tails in our distributions, we proceed in two steps. First, we graphically analyze the tail behavior of the sample distributions. Second, we formally estimate the index of stability.
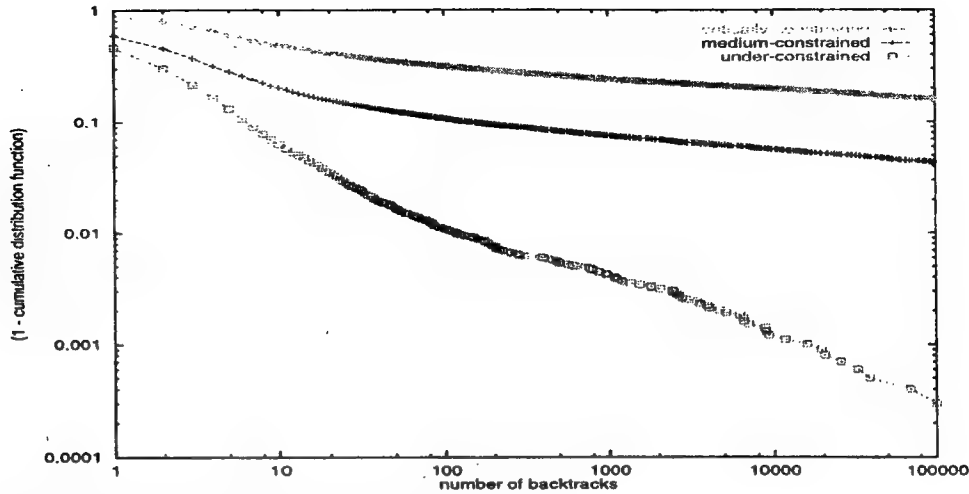
Figure 6b: Log-log plot of heavy-tailed behavior.

If a Pareto-Lévy tail is observed, then the rate of decrease of the estimated density is hyperbolic — *i.e.*, slower than the exponential rate. The complement to one of the cumulative distribution also displays a hyperbolic decay

$$1 - F(x) = \Pr\{X > x\} \sim C.x^{-\alpha}. \tag{4}$$

Then, for an heavy-tailed random variable, a log-log plot of the frequency of observed backtracks after x should show an approximate linear decrease at the tail. Moreover, the slope of the observed linear decrease provides an estimate of the index α. In contrast, for a distribution with an exponentially decreasing tail, the log-log plot should show a faster-than-linear decrease of the tail.

Since the described behavior is a property of the tail we should mainly be concerned with the last observations, say the 10% observations that display a higher number of backtracks.

In Figure 6b, we have plotted three empirical cumulative distributions. One based on the probability distribution from Figure 6a (under-constrained), another for a medium constrained (solvable) instance, and a third for a critically constrained (solvable) instance. The linear nature of the tails in this log-log plot directly reveals tails of the Pareto-Lévy type.

For contrast we show in Figure 6c the log-log plots of two standard probability distributions. We see sharp rounded drop-off of both curves — indicating the absence of heavy tails. The distributions are given by the cost profiles on two unsolvable instances of our quasigroup completion problem. One is a rare unsolvable problem in the underconstrained area (best fit: a gamma distribution), the other is an unsolvable instance in the critically constrained region (best fit: normal distribution).

To complement our visual check of Figure 6b, and obtain an estimate of the index of stability (the value of α), we use the method of Hall (1982), which
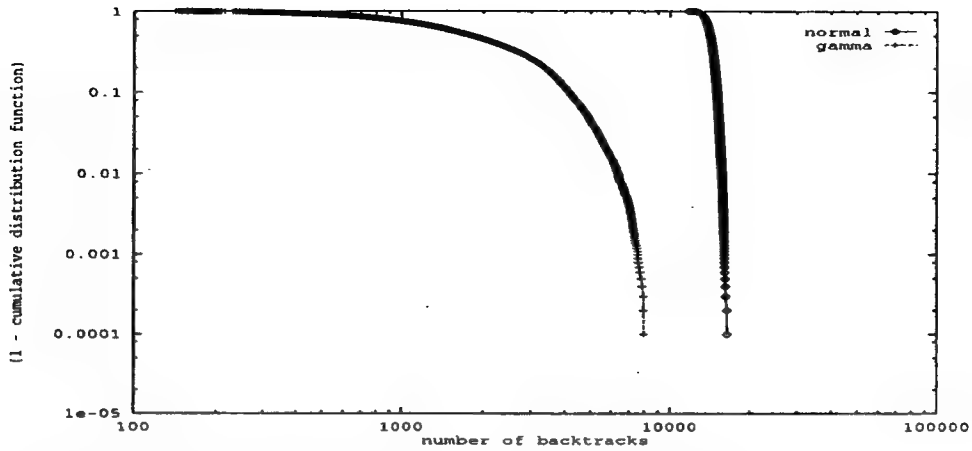
Figure 6c: Log-log plot of standard distributions (no heavy tails).

performs a regression on the extreme tails. Let $X_{n1} \leq X_{n2} \leq \ldots \leq X_{nn}$ be the order statistics, i.e., the ordered values of the sample $X_1, X_2 \ldots, X_n$ of the obtained number of backtracks. Set $r < n$ as a truncation value which allows us to consider only the extreme observations. We obtain the estimator

$$\hat{\alpha}_r = \left(r^{-1} \sum_{j=1}^{r} \log X_{n,n-j+1} - \log X_{n,n-r} \right)^{-1} \qquad (5)$$

This is a maximum likelihood estimator and Hall (1982) has established its asymptotic normality. Hall has also determined the optimal choice of the truncation parameter r. However, since this parameter is a function of the *unknown* parameters of the distribution, we adhere here to the common practice of using a set of values in the range $\{n/10, n/25\}$. This corresponds to severe truncations, which allow us to be more confident in our results.

We examined over two dozen distributions, and found values for α that are consistent with the infinite variance hypothesis ($\alpha < 2$) and, in many cases, they point to the nonexistence of the mean ($\alpha < 1$). The estimates of α for the distributions in Figure 6b were consistent with the hypothesis of infinite variance and infinite mean. The standard deviation in the estimates of the α values were consistently an order of magnitude smaller than the estimates themselves, pointing to highly significant coefficients.

Are heavy-tailed distributions able to explain the strange sample mean discussed in the introduction? In other words, are stable distributions with index of stability of the order of magnitude of those estimated, able to generate data which reproduces the pattern shown in Figure 1a?

By using the method of Chambers, Mallows, and Stuck (1976), we generated random samples from a stable distribution, and calculated the mean as function

27
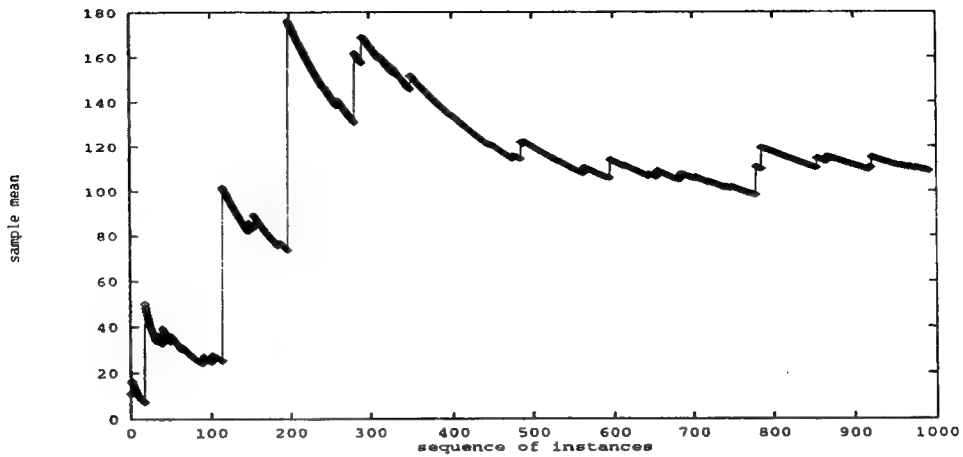
Figure 7: Behavior of mean for example stable distribution.

of the number of samples. The resulting sequence of partial means is portrayed on Figure 7. The comparison between Figures 1a and 7 is striking, as the general wild oscillations are very similar and characteristic of heavy-tailed distributions.

5   Exploiting Heavy-Tailed Behavior

Figure 8a: Using restarts to exploit heavy-tailed behavior.

For our heavy-tailed distributions, we see that our procedures are in some sense most effective early on in the search. This suggests that a sequence of short

Figure 8b: Log-log plot for restarts.

runs instead of a single long run may be a more effective use of our computational resources. We explore this idea by cons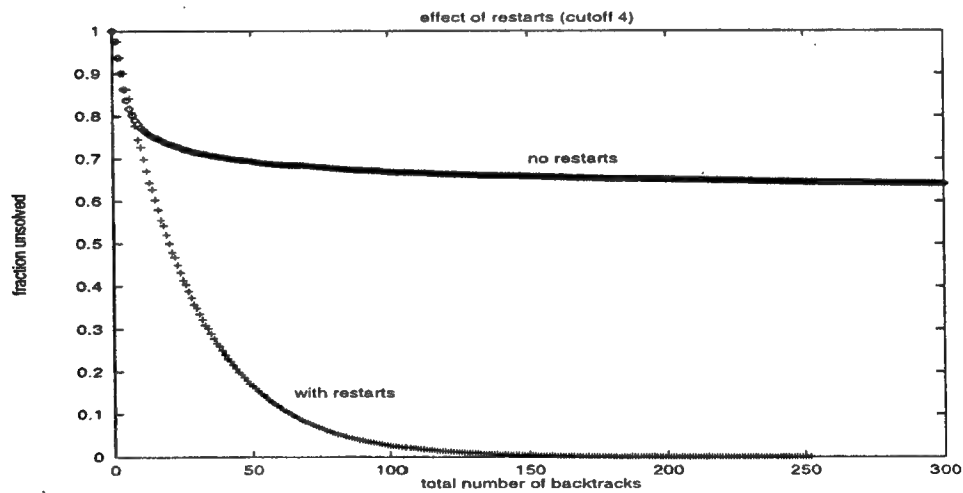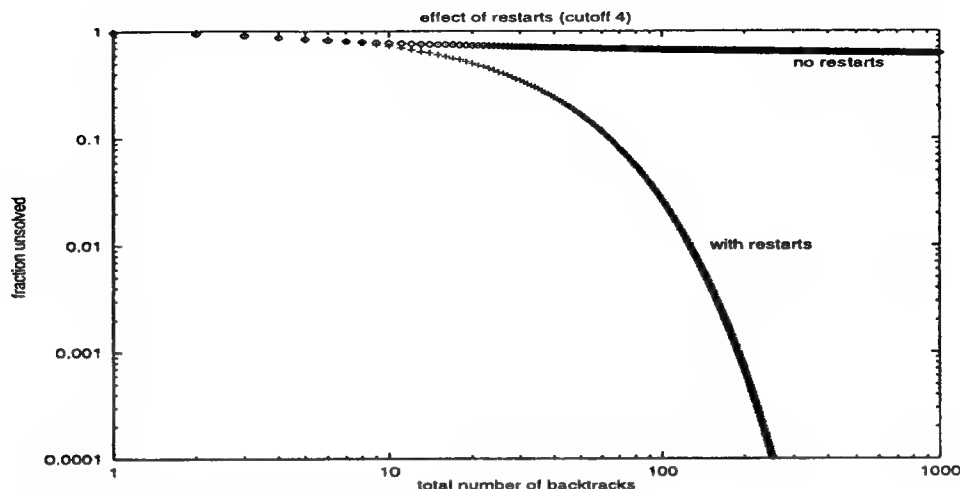idering a fixed limit L on our overall cost ("run time"). From the cumulative cost distribution and L, we can determine what our expected probability of *not* solving the instance is because the search procedure runs out of time. We can also compute this "probability of failure" for a procedure that quickly restarts. Figures 8a and 8b give the results of such an analysis. (For more detailed results on the derivations of the probability distributions for restarts, see Gomes and Selman, Rome Lab Technical Report, 1997. For related work, see Gomes and Selman 1997b.)

The analysis was done for the completion problem of an instance of order 20 with 5% pre-placed. See distribution in Figure 6a. From Figure 8a, we see that without restarts and given a total of 50 backtracks, we have a failure rate of around 70%. Using restarts (every 4 backtracks), this failure rate drops to around 10%. With an overall limit of only 150 backtracks, the restart startegy solves the instance almost always, whereas the original procedure still has a failure rate of around 70%. Such a dramatic improvement due to restarts is typical for heavy tailed distributions — in particular, we get similar results on critically constrained instances. Finally, Figure 8b shows a clear downward curve for the restart strategy. This suggests that the heavy-tailed nature of the cost distribution has disappeared. And, thus, we see that random restarts provide an effective mechanism for dealing with heavy-tailed cost distributions. These results explain the informal popularity of restart strategies in combinatorial search methods.

6   Conclusions and Future Work

We have revealed the special heavy-tailed nature of the cost distribution of combinatorial search procedures. We showed how such distributions can be modeled

as stable distributions with heavy Pareto-Lévy type tails. Our analysis explains the empirically observed erratic behavior of the mean and variance of the cost of combinatorial search. And, more generally, the high variability observed between runs of such procedures.

Stable distributions have recently been used to capture a variety of real-world phenomena, such as stock market and wheather patterns. We believe our results are the first indication of the occurrence of such distributions in purely computational processes. We hope that our results will further stimulate research along these lines by employing the special statistical tools available in this area.

We also showed how a "restart" strategy is an effective remedy against the heavy-tailed phenomena. Restarts drastically reduce the probability of failure under limited time resources and reduce the overall variability of the method. Of course, when heavy tails are absent, restarts are much less effective. In our study, we did not encounter heavy tails for unsolvable instances.

References

Anderson, L. (1985). Completing Partial Latin Squares. *Mathematisk Fysiske Meddelelser*, 41, 1985, 23–69.

Brelaz, D. (1979). New methods to color the vertices of a graph. *Comm. of the ACM* (1979) 251–256.

Chambers, John M., Mallows, C.L., and Stuck, B.W. (1976) A method for simulating stable random variables. *Journal of the American Statistical Association* **71**, 340–344.

Cheeseman, Peter and Kanefsky, Bob and Taylor, William M. (1991). Where the Really Hard Problems Are. *Proceedings IJCAI-91*, 1991, 163–169.

Colbourn, C. (1983). Embedding Partial Steiner Triple Systems is NP-Complete. *J. Combin. Theory* (A) 35 (1983), 100-105.

Dechter, R. (1991) Constraint networks. *Encyclopedia of Artificial Intelligence* John Wiley, New York (1991) 276-285.

de Lima, Pedro J.F. (1997). On the robustness of nonlinearity tests to moment condition failure. *Journal of Econometrics* **76**, 251–280.

Denes, J. and Keedwell, A. (1974) Latin Squares and their Applications. *Akademiai Kiado, Budapest, and English Universities Press*, London, 1974.

Frost, Daniel , Rish, Irina, and Vila, Lluís (1997) Summarizing CSP hardness with continuous probability distributions. *Proc. AAAI-97*.

Fujita, M., Slaney, J., and Bennett, F. (1993). Automatic Generation of Some Results in Finite Algebra *Proc. IJCAI*, 1993.

Freuder, E. and Mackworth, A. (Eds.). *Constraint-based reasoning.* MIT Press, Cambridge, MA, USA, 1994.

Gent, I. P. and Walsh, T.. (1993). Easy Problems are Sometimes Hard, the DIMACS Challenge on Satisfiability Testing. Piscataway, NJ, Oct. 1993. Full version AIJ (Hogg et al. 1996).

Gent, I. and Walsh, T. (1996) The Satisfiability Constraint Gap. *Artificial Intelligence*, 81, 1996.

Gomes, C.P. and Selman, B. (1997a) Problem Structure in the Presence of Perturbations *Proc. AAAI-97*, Providence, RI, 1997.

Gomes, C.P. and Selman, B. (1997b) Algorithm Portfolio Design: Theory vs. Practice, *Proc. UAI-97*, Providence, RI, 1997.

Hall, Peter (1982) On some simple estimates of an exponent of regular variation. *Journal of the Royal Statistical Society*, **B 44**, 37–42.

Huberman, B.A., Lukose. R.M., and Hogg, T. (1997). An economics approach to hard computational problems. *Science*, 265, 51–54.

Hogg, T., Huberman, B.A., and Williams, C.P. (Eds.) (1996). Phase Transitions and Complexity. *Artificial Intelligence*, 81 (Spec. Issue; 1996)

Kirkpatrick, S. and Selman, B. (1994) Critical Behavior in the Satisfiability of Random Boolean Expressions. *Science*, 264 (May 1994) 1297–1301.

Kwan, Alvin C. M. (1995) Validity of normality assumption in CSP research, *PRICAI'96: Topics in Artificial Intelligence. Proceedings of the 4th Pacific Rim International Conference on Artificial Intelligence*, 459–465.

Lam, C., Thiel, L., and Swiercz, S. (1989) The Non-existence of Finite Projective Planes of Order 10. *Can. J. Math.*, Vol. XLI, 6, 1989, 1117-1123.

Mandelbrot, Benoit B. (1960) The Pareto-Lévy law and the distribution of income. *International Economic Review* 1, 79–106.

Mandelbrot, Benoit B. (1963) The variation of certain speculative prices. *Journal of Business* **36**, 394–419.

Mandelbrot, B. (1983) *The fractal geometry of nature*. Freeman: New York. 1983.

Mitchell, D., Selman, B.. and Levesque, H.J. (1989) Hard and easy distributions of SAT problems. *Proc. AAAI-92*, San Jose, CA (1992) 459–465.

Puget, J.-F. (1994) A C++ Implementation of CLP. *Technical Report 94-01* ILOG S.A., Gentilly, France, (1994).

Russell, S and Norvig P. (1995) *Artificial Intelligence a Modern Approach*. Prentice Hall, Englewood Cliffs, NJ. (1995).

Samorodnitsky, Gennady and Taqqu, Murad S. (1994) *Stable Non-Gaussian Random Processes: Stochastic Models with Infinite Variance*, Chapman and Hall, New York.

Selman, B. and Kirkpatrick, S. (1996) Finite-Size Scaling of the Computational Cost of Systematic Search. *Artificial Intelligence*, Vol. 81, 1996, 273–295.

Smith, B. and Dyer, M. Locating the Phase Transition in Binary Constraint Satisfaction Problems. *Artificial Intelligence*, 81, 1996.

Smith, B. and Grant S.A., Sparse Constraint Graphs and Exceptionally Hard Problems. *IJCAI-95*, 646–651, 1995. Full version in AIJ (Hogg et al. 1996).

van Hentenryck, P. , Deville, Y., and Teng Choh-Man (1992) A generic arc consistency algorithm and its specializations. *Artificial Intelligence*, 57, 1992.

Williams, C.P. and Hogg, T. (1992) Using deep structure to locate hard problems. *Proc. AAAI-92*, San Jose, CA, July 1992, 472–277.

Zhang, W. and Korf, R. A Study of Complexity Transitions on the Asymmetric Travelling Salesman Problem. *Artificial Intelligence*, 81, 1996.

Zolotarev, V.M. (1986) One-dimensional Stable Distributions. Vol. 65 of "Translations of mathematical monographs", American Mathematical Society. Translation from the original 1983 Russian Ed.

Boosting Combinatorial Search Through Randomization

Carla P. Gomes
Computer Science Department
Cornell University
Ithaca, NY 14853
gomes@cs.cornell.edu

Bart Selman
Computer Science Department
Cornell University
Ithaca, NY 14853
selman@cs.cornell.edu ·

Henry Kautz
AT&T Labs
180 Park Avenue
Florham Park, NJ 07932
kautz@research.att.com

Abstract

Unpredictability in the running time of complete search procedures can often be explained by the phenomenon of "heavy-tailed cost distributions", meaning that at any time during the experiment there is a non-negligible probability of hitting a problem that requires exponentially more time to solve than any that has been encountered before (Gomes *et al.* 1998a). We present a general method for introducing controlled randomization into complete search algorithms. The "boosted" search methods provably eliminate heavy-tails to the right of the median. Furthermore, they can take advantage of heavy-tails to the left of the median (that is, a non-negligible chance of very short runs) to dramatically shorten the solution time. We demonstrate speedups of several orders of magnitude for state-of-the-art complete search procedures running on hard, real-world problems.

Introduction

The time required by complete search methods to solve similar combinatorial problems can be surprisingly variable. Two problem instances may be identical, except for the order in which the variables are numbered. A particular complete search algorithm may solve the first in seconds, yet require hours or days to solve the second. Even for a single problem instance, a seemingly trivial change in a detail of the search algorithm may drastically alter the solution time. In addition, in many domains there are problem instances that can be quickly solved by incomplete methods, but apparently cannot be solved by complete methods, even methods guided by powerful heuristics.

This unpredictability in the running time of a complete algorithm undermines one of the main reasons that one may choose to employ such a method, namely the desire for a guarantee that the algorithm will determine whether or not each problem instance in fact has a solution. It is desirable, therefore, to find ways to improve the robustness and predictability of these algorithms.

This paper discusses a general technique for improving complete search methods by introducing a controlled amount of *randomization*. The technique actually takes advantage of the variability of the underlying search method in order to find solutions more quickly and with less variance in solution time. We demonstrate the effectiveness of this strategy on SAT and CSP algorithms, in the domains of logistics planning, circuit synthesis, and round-robin scheduling. Solutions times are reduced by an order of magnitude or more, and some instances are solved for the first time by a method other than local search.

We will show that the unpredictability in running times for combinatorial algorithms can often be explained by a phenomenon called a "heavy-tailed cost distribution" (Gomes *et al.* 1998a). In our preliminary experiments, we plotted the solution times for a deterministic search algorithm running on a random distribution of scheduling problem instances. We noticed that at any time during the experiment there was a non-negligible probability of hitting a problem that required exponentially more time to solve than any that had been encountered before. This so-called "heavy-tail" phenomena causes the mean solution time to increase with the length of the experiment, and to be infinite in the limit.

Previous authors have noted the occurrence of seemingly exceptionally hard problems in fixed problem distributions (Gent and Walsh 1994; Smith and Grant 1996). However, we further discovered that when a small amount of randomization was introduced into the heuristic used by the search algorithm, then, on some runs, the instances were solved quickly. Thus, the "hardness" did not reside in the instances, but rather in the combination of the instance with the details of the deterministic algorithm. When we plotted the solution times for many runs of the randomized complete algorithm (with different random seeds) on a *single* problem instance, we discovered the same heavy-tailed distribution as we had seen before on a collection of instances.

This observation led us to realize that a deterministic search algorithm can be viewed as a *single run* of a randomized algorithm. The unpredictability of deterministic, complete algorithms is thus explained by the variance one would expect in any one run of a randomized algorithm. Furthermore, by analyzing the shape of the cost distribution we developed simple techniques that provably *reduce*

Problem	Solver	Deterministic soln. time	Randomized mean soln. time
logistics.d	Satz	108 min	95 sec
3bit-adder-32	Satz	> 24 hrs	165 sec
3bit-adder-31	Satz	> 24 hrs	17 min
round-robin 14	ILOG	411 sec	250 sec
round-robin 16	ILOG	> 24 hrs	1.4 hrs
round-robin 18	ILOG	> 48 hrs	≈ 22 hrs
block-world.d	Satz	30 min	23 min

Table 1: Comparison of speed of original deterministic algorithms and randomized versions on test-bed problems.

the mean solution time.

For our experiments, we used known hard problem instances from scheduling, planning, and circuit synthesis, and a state of-the-art satisfiability engine (Satz, by Li and Anbulagan (1997)), and a highly efficient CSP solver built using the ILOG C++ constraint programming library (Puget and Leconte 1995). It is important to note that in both cases the underlying deterministic complete search engines are among the fastest (and on many problems, *the* fastest) in their class. Thus, the techniques discussed in this paper extend the range of complete methods to problems that were previously beyond their reach. For a preview of our main results, see Table 1. The table shows how our randomization strategy enabled us to solve several previously unsolved problem instances, and other instances were solved up to an order of magnitude faster. Given the techniques' simplicity and generality, our approach can be easily adapted to improve the performance of other backtrack-style search methods used in planning, scheduling, and other tasks of interest to AI.

Problem Domains

Our problem domains are timetable scheduling, planning, and circuit synthesis. The first is formalized as a CSP problem, and the latter two as propositional satisfiability.

Timetabling consists in determining whether there exists a feasible schedule that takes into consideration a set of pairing and distribution constraints. More specifically, we consider problems derived from sports scheduling applications. The literature in this area is growing, and one can begin to get a sense of the range and mathematical difficulty of the problems encountered (McAloon *et al.* 1997; Nemhauser and Trick 1997; and Schreuder 1992). Here we consider the timetabling problem for the classic "round-robin" schedule: every team must play every other team exactly once. The problem is formally defined as follows:

1. There are N teams (N even) and every two teams play each other exactly once.
2. The season lasts $N - 1$ weeks.

3. Every team plays one game in each week of the season.
4. There are $N/2$ periods and, each week, every period is scheduled for one game.
5. No team plays more than twice in the same period over the course of the season.

Up to 8-team problems are relatively simple and can be done by brute force. However, the combinatorics of this scheduling problem are explosive. For an N team league, there are $N/2 \cdot (N-1)$ matchups (i, j) with $0 \leq i < j < N$ to be played. A schedule can be thought of as a permutation of these matchups. So, for N teams the search space size is $(N/2 \cdot (N-1))!$, *i.e.*, the search space size grows as the factorial of the square of $N/2$. Published algorithms for this problem all scale poorly, and the times for our *deterministic* solver (as shown in Table 1) are among the best (see also Gomes *et al.* 1998b).

The second domain is planning. Kautz and Selman (1996) showed that propositional SAT encodings of difficult STRIPS-style planning problems could be efficiently solved by SAT engines. While both a complete backtrack-style engine and an incomplete local-search engine worked well on moderate-sized problems, the largest problems from the domain of logistics scheduling could only be solved by local search. However, it turns out that the deterministic version of Satz can solve all of the logistics instances from that paper in less than 2 minutes. Therefore we constructed a still-larger planning problem, labeled "logistics.d". This domain involves moving packages on trucks and airplanes between different locations in different cities. While the largest logistics problem from the Kautz and Selman (1996) paper involved 1,141 variables, "logistics.d" involves 2,160 variables.

The final domain is circuit synthesis. Kamath *et al.* (1993) developed a technique for expressing the problem of synthesizing a programmable logic array (PLA) as a propositional satisfiable problem. The statement of the problem includes a table specifying the function to be computed, and an upper-bound on the number of gates that may appear in the circuit. In general, these problems become more difficult to solve as the number of gates is reduced, until the

limit is reached where the instance is unsatisfiable. These problems are quite hard to solve with complete SAT procedures, and have been used as part of the test-beds for numerous SAT competitions and research studies. The problems considered in this paper, "3bit-adder-32" and "3bit-adder-31" are (as one would guess) based on synthesizing a 3-bit adder using 32 and 31 gates respectively. Although Selman and Kautz (1993) solve the instances using local search, no one has previously solved either using a backtrack-style procedure.

Randomizing Complete Search Engines

We now consider general techniques for adding randomization to complete, systematic, backtrack-style search procedures. Such a procedure constructs a solution incrementally. At each step a heuristic is used to select an operation to be applied to a partial solution, such as assigning a value to an unassigned variable. Eventually either a complete solution is found, or the algorithm determines that the current partial solution is inconsistent. In the latter case, the algorithm backtracks to an earlier point in its search tree.

If several choices are heuristically determined to be equally good, then a deterministic algorithm applies some fixed rule to pick one of the operations; for example, to select the lowest-numbered variable to assign. The most obvious place to apply randomization, therefore, is in this step of tie-breaking: if several choices are ranked equally, choose among them at random. Even this simple modification can dramatically change the behavior of a search algorithm, as we will see in the section on CSP below.

However, if the heuristic function is particular powerful, it may rarely assign more than one choice the highest score. To handle this, we can introduce a "heuristic equivalence" parameter to the algorithm. Setting the parameter to a value H greater than zero means all choices who receive scores within H-percent of the highest score are considered equally good. This expands the choice-set for random tie-breaking.

With these changes each run of the search algorithm on a particular instance will differ in the order in which choices are made and potentially in time to solution. As we will discuss in detail below, it can be advantageous to terminate searches which appear to be "stuck", exploring a part of the space far from a solution. Therefore we will also introduce a "cutoff" parameter, that limits search to a specified number of backtracks. When the cutoff is reached, the algorithm is restarted at the root of the search tree.

We should note that introducing randomness in the branching variable selection does not effect the completeness of the backtrack-style search. Some basic bookkeeping (only linear space) ensures that the procedures do not revisit any previously explored part of the search space, which means that we can still determine inconsistencies,

unlike local search methods. The cutoff parameter does limit the size of the space that can be searched exhaustively between restarts. In practice, we gradually increase the cutoff, to allow us to determine inconsistencies, if necessary.

A variable-order randomization and restart strategy was employed in Crawford and Baker's (1994) "probing" algorithm for SAT. Despite the fact that it performed no backtracking at all, it was shown to solve a number of examples. Even though, the "power of randomization" in combinatorial search has been informally recognized by others (for recent work in scheduling domains, see e.g., Bresina 1996 and Oddi and Smith 1997), our work provides the first explanation for the potential success of this kind of strategy, in terms of heavy-tailed distributions (Gomes *et al.* 1998a). As we will see, our data also shows that there is often a clear optimal cutoff value; simply probing down with unit propagation but no backtracking can be ineffective. For example, in Table 3 we have a 0% success rate for a cutoff value of 2. More recently, Bayardo and Schrag (1997) introduced a backtrack-style solver, rel-sat, that included randomized tie-breaking and restarts, but with only a fixed, high cutoff value. The focus of that work was on the backtracking technique, rather than the effect of restarts.

The first complete search algorithm we randomized was a CSP solver. ILOG SOLVER is a powerful C++ constraint programming library (Puget and Leconte 1995). For the round-robin scheduling problems discussed below, we used the library to build a deterministic, backtrack-style CSP engine. (See Dechter (1991) and Freuder and Mackworth (1994) for an overview of basic CSP algorithms.) It employs the first-fail heuristic for variable assignment, which selects the variables with the smallest domain first; ties are broken lexicographically. The performance of this deterministic version already matches or exceeds all the published results on solving these types of problems. We then randomized the solver by breaking ties randomly, and adding a cutoff parameter (Gomes *et al.* 1998b).

The second algorithm we randomized was for propositional satisfiability. One of the fastest complete search engines for propositional satisfiability testing is the Satz system of Li and Anbulagan (1997). Satz is a version of the Davis-Putnam-Loveland procedure (Davis *et al.* 1962), with a heuristic based on choosing a branch variable that maximizes a function of the number of the unit propagations performed when it is set positively or negatively. Satz is the fastest deterministic SAT procedure we have found for the instances discussed in this paper. It can often solve smaller instances of these types with less than 100 backtracks. Because its heuristic usually chooses a single branching variable without ties, we added a heuristic equivalence parameter to enlarge the choice-set.

Heavy-Tailed Cost Distributions

In previous work (Gomes *et al.* 1998a), we show that the tail behavior of randomized complete backtrack style methods is often best modeled using distributions which asymptotically have tails of the Pareto-Lévy form, *viz.*

$$\Pr\{X > x\} \sim C.x^{-\alpha}, \quad x > 0 \qquad (1)$$

where $\alpha > 0$ is a constant (Mandelbrot 1960; and Samorodnitsky 1994). These are heavy-tailed distributions, *i.e.*, distributions whose tails have a *power law decay*. The constant α is called the *index of stability* of the distribution. For $\alpha < 2$, moments of X of order less than α are finite while all higher order moments are infinite, *i.e.*, $\alpha = \sup\{a > 0 : E|X|^a < \infty\}$. For example, when $\alpha = 1.5$, the distribution has a finite mean but no finite variance. With $\alpha = 0.6$, the distribution has neither a finite mean nor a finite variance.

If a Pareto-Lévy tail is observed, then the rate of decrease of the distribution is a power law. (Standard distributions exhibit exponential decay.) From (1), we have $1 - F(x) = \Pr\{X > x\} \sim C.x^{-\alpha}$, so the complement-to-one of the cumulative distribution, $F(x)$, also decays according to a power law. Given the power law decay of the complement-to-one of the cumulative distribution of a heavy-tailed random variable, its log-log plot should show an approximately linear decrease in the tail. Moreover, the slope of the observed linear decrease provides an estimate of the index α.
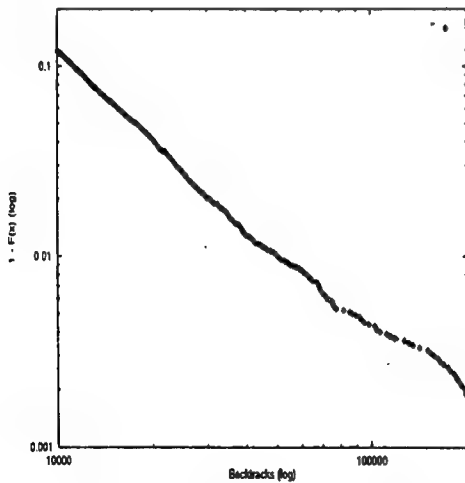

Figure 1: Log-log plot of the tail of 12 team round-robin scheduling.

Figure 1 shows the log-log plot of the tail ($X > 10,000$) of the complement-to-one of the cumulative distribution, $1-F(x)$, for our 12 team round-robin problem. The linear nature of the tail in this plot directly reveals heavy-tails of the Pareto-Lévy type.

To complement our visual check of heavy-tailed behavior of Figure 1, we calculate the maximum likelihood estimate of the index of stability (the value of α): For our round-robin scheduling problem, for $N = 12$, we obtain $\alpha = 0.7$, which is consistent with the hypothesis of infinite mean and infinite variance, since $\alpha \leq 1$.[1]

So far, we have identified heavy-tailed behavior of the cost distribution to the right of the median. The heavy tail nature shows that there is a computationally significant fraction of very long runs, decaying only at a polynomial rate. The strategy of running the search procedure with a cutoff near the median value of the distribution clearly avoids these long runs in the tail.

However, our experiments in Gomes (1998a) also suggest a heavy tail phenomenon on the left-hand side of the median value of the cost distribution, which means that the success rate for a solution only increases polynomially with the number of backtracks. This explains how a relatively low cutoff value still gives a sufficiently high success rate to allow us to solve a problem instance. For example, for our round-robin scheduling problems with $N = 16$, we observed several runs that took less than 200 backtracks, compared to a median value of around 2,000,000. For $N = 18$, we ran with a cutoff of $500,000$ and solved the instance after 20 tries. Each try took about 1 hour, and the successful run took 350,632 backtracks.

Tails on the left are also characterized by an index of stability. Based on our data (Gomes 1998a), we conjecture that α for the tail on the left is less than 1.0 on hard combinatorial search problems. This conjecture has strong implications in terms of algorithm design: It means that in order to obtain the minimal expected run time, a preferred strategy consists of relatively short runs of a randomized backtrack-style procedure.

We do not wish to give the impression that *every* search problem gives rise to a heavy-tailed distribution. In fact, doing so would give rise to the suspicion that the distributions we found were an artifact of our methodology, rather than a real phenomena of the problem domain! One domain in which we have *not* found heavy-tails is on blocks-world planning problems. The hardest blocks-world problem from Kautz and Selman (1996) is blocks-world.d, and it can be solved by deterministic Satz in 30 minutes. We ran the randomized version of Satz on this instance at a wide range of cutoff values and heuristic equivalence settings. The optimal equivalence parameter setting was 30%. However, over a range of cutoff values, there was no evidence of a heavy-tailed distribution, and, therefore, randomization

---

[1] Of course, the computational cost of complete backtrack-style algorithms has a finite upper-bound. However, since we are dealing with NP-complete problems, this upper-bound is exponential in the size of the problem, which means that *de facto*, for realistic-size hard instances, it can be treated as infinite for practical purposes: no practical procedure can explore the search full space.

only slightly increases the effectiveness of Satz: the mean cost is 23 minutes. Further studies are needed to determine exactly what characteristics of combinatorial search problems lead to heavy-tailed behavior.

Boosting Performance by Randomization and Restarts

So far, we have discussed how heavy-tailed probability distributions underlie the large variability observed when running a randomized backtrack-style procedure on a variety of problem instances. We can obtain more efficient and more predictable procedures by running the search up to a certain cutoff point and then restarting at the root of the tree. Restarts clearly prevent the procedure from getting trapped in the long tails on the right of the distribution. In addition, a very low cutoff value can also be used to exploit the heavy-tails to the left of the median, and will allow us to solve previously unsolved problem instances after a sufficient number of restarts. In Table 1, the mean solution times in the "Randomized" column are based on empirically determined near-optimal cutoff values. For each randomized solution time the standard deviation is of the same order of magnitude as the mean. This is to be expected because the distribution is geometric, as will be shown in the next section. Without restarts, of course, the variance and mean tend to infinity to a first approximation.

We will now discuss these results in more detail.

Our deterministic CSP procedure on the round-robin scheduling problem gives us us a solution for $N = 14$ in about 411 seconds. (Experiments ran on a 200MHz SGI Challenge.) We could not find a solution for $N = 16$ and $N = 18$. Apparently, the problem quickly becomes very difficult, even for moderate values of N. The subtle interaction between global and local constraints makes the search for a globally consistent solution surprisingly hard.

For problems for which we can empirically determine the overall cost profile, we can calculate an *optimal* cutoff value to minimize the expected cost of finding a solution. Our main interest, however, is in solving previously unsolved instances, such as the $N = 16$ and $N = 18$ case. These problems are too hard to obtain a full cost distribution. For example, for $N = 16$, running with a cutoff of 1,000,000 gives a success rate of less than 40%, so we do not even reach the median point of the distribution. Each run takes about 2 hours to complete. (We estimate that the median value is around 2,000,000. Our deterministic procedure apparently results in a run that still lies to the right of the expected median cost.) In order to find a good cutoff value for very hard problem instances, the best available strategy is a trial-and-error process, where one experiments with various cutoff values, starting at relatively low values, since the optimal cutoff for these problems tends to lie below the median value of the distribution. This can be seen from Table

cutoff	succ. rate	mean cost ($\times 10^6$)
200	0.0001	2.2
5,000	0.003	1.5
10,000	0.009	1.1
50,000	0.07	0.7
100,000	0.06	1.6
250,000	0.21	1.2
1,000,000	0.39	2.5

Table 2: Solving the 16-team robin-robin scheduling problem for a range of cutoff values.

cutoff	succ. rate	mean cost
2	0.0	>300,000
4	0.00003	147,816
8	0.0016	5,509
16	0.009	1,861
32	0.014	2,405
250	0.018	13,456
16000	0.14	107,611
128000	0.32	307,550

Table 3: Solving the logistics.d problem for a range of cutoff values.

2, which gives the expected cost (backtracks) for finding a solution for $N = 16$ for a range of cutoff values. The optimal cutoff is around 5.10^4, resulting in an expected cost per solution of 7.10^5 backtracks ($\approx$ 1.4 hrs). For the $N = 18$ case, we ran with a cutoff of 5.10^5, and found a solution after approximately 22 hours.[2]

Table 3 gives the performance of Satz for a range of cutoff values on the logistics.d instance. Again, there is a clear optimal value: In this case, it's surprisingly low, 16 backtracks. Despite the low success rate (less than 1%) at this cutoff value, the overall performance is close to optimal here, requiring around 1,800 backtracks total per solution, which takes around 95 seconds. Compare this with the 108 minutes for the deterministic version of Satz. It's important to note that the 108 minutes run is not just an "unlucky" determinist run. Given the shape of the underlying heavy-tailed distribution, most runs take more than 100,000 backtracks (over 1 hour). The trick is to exploit the fact that we have a non-negligible probably of solving the instance in a *very* short run. Our fast restart strategy exploits this.

See Table 1 for other improvements due to randomiza-

---

[2]Since the submission of this paper, a lot of progress has been made in terms of solving larger instances (McAloon *et al.* in preparation). By using multiple threads on a 14 processor Sun system, 26 and 28 teams schedules were generated, which is the record as of this writing (Wetzel and Zabatta, 1998). We believe these numbers can be improved upon with our randomization technique.

tion. Until now, the 3bit-adder problems had not been solved by any backtrack-style procedure. On the block-world problem, we obtain little improvement, which can be attributed to the absence of heavy-tails as discussed above.

These results show that introducing a stochastic element into a backtrack-style search procedure, combined with an appropriate restart strategy, can significantly enhance the procedure's performance. In fact, as we see here, it allows us to solve previously unsolved problem instances.

A Formal Analysis of Restarts

In this section we formalize the strategy of restarts S of a complete stochastic procedure A. We derive the probability distribution of S assuming the full knowledge of the probability distribution of A. We demonstrate that the probability distribution associated with S does not exhibit heavy tails. Furthermore, S has a finite mean and variance, even if the stochastic procedure A has an infinite mean and variance.

Let us consider a complete stochastic procedure and associate with it the random variable A, where A is the number of backtracks that it takes to find a solution or prove that it does not exist. Let us now consider the following stochastic strategy for running A: run A for a fixed number of backtracks c (the cutoff); if A finds a solution or proves it does not exist, then our stochastic strategy has also found a solution (or proved that it does not exist) and it stops. Otherwise, restart A from the beginning, using an independent random seed, for another c backtracks, and so on. Define S as the number of backtracks that the stochastic strategy of restarts of A with cutoff c takes to find a solution or prove that it does not exist. Let's assume that we know $P[A \leq c]$, *i.e.*, the probability that the stochastic procedure A will find a solution or prove that it does not exist in no more than c backtracks. The sequence of runs of A executed by our restart strategy are independent, and therefore they can be seen as a sequence of Bernoulli trials, in which the success consists in finding a solution (or proving that it doesn't exist) before the end of the run.

It's convenient to also define a random variable R, giving the number of restarts until a solution is found (or the instance is shown inconsistent). Note that $R = \lceil S/c \rceil$. R follows a *geometric distribution* with parameter $p = P[A \leq c]$. The probability of the tail of S, $P[S > s]$, is given by

$$P[S > s] = (1 - p)^{\lfloor s/c \rfloor} P[A > s \bmod c]$$

Taking into consideration that $R = \lceil S/c \rceil$ and that it follows a geometric distribution (exponential decay; finite mean and variance), it follows that the tail of the distribution of S also exhibits exponential decay and S has a finite mean and variance.

We should emphasize that when adopting a low cutoff the strategy of restarts partially eliminates the heavy tail on the left: the lower the cutoff, the shorter the tail. This is true since the distribution of S exhibits exponential decay for $S >$ cutoff.

Conclusions

Building on our previous work on heavy-tailed behavior in combinatorial search (Gomes et al. 1998a), we have shown that performance of complete, backtrack-style search algorithms on hard real-world problems can be greatly enhanced by the addition of randomization combined with a rapid restart strategy. Speedups of several orders of magnitude were observed, and some test problem instances were solved for the first time by any backtrack-style procedure.

The success of our approach is based on exploiting the heavy-tailed nature of the cost distributions. We saw that in most of the domains we found that "outliers" on *both* sides of the median occur with a relatively high frequency. Heavy-tails to the right of the median cause the mean solution time to grow without bounds. Adding cutoffs and restarts to the search algorithm, however, both theoretically and empirically eliminate the heavy-tail and bound the mean. Heavy-tails to the left of the mean can be exploited by performing many rapid restarts with short runs, leading to a further dramatic decrease in expected solution time.

We applied the randomization techniques to two state-of-the-art search engines for CSP and propositional satisfiability. We were able to solve hard round-robin scheduling instances of up to size 18, when the corresponding deterministic version could only handle instances up to size 14. In the domain of planning as satisfiability, we extended the range of logistics problems that could be solved by complete methods from problems containing 1,141 variables to ones involving 2,160 variables (solved with mean cost of 95 seconds).

It would be interesting to explore our randomization approach in context of other backtrack-style approaches, such as dynamic backtracking (Ginsberg 1993). We believe that the generality of the approach will lead to further advances in planning, scheduling, diagnosis, game-playing, and other areas of AI.

References

Alt, H., Guibas, L., Mehlhorn, K., Karp, R., and Wigderson A. (1996). A method for obtaining randomized algorithms with small tail probabilities. *Algorithmica*, 16, 1996, 543–547.

Bayardo, Roberto J., and Schrag, Robert C. (1997). Using CSP look-back techniques to solve real-world SAT instances. *Proc. AAAI-97*, New Providence, RI, 1997, 203–208.

Bresina, J. (1996) Heuristic-biased stochastic sampling. *Proc. AAAI-96*, Portland, OR, 1996.

Crawford, J. M., and Baker, A. B. (1994). Experimental results on the application of satisfiability algorithms to scheduling problems. *Proc. AAAI-94*, Seattle, WA, 1092–1097.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem proving. *Comm. ACM*, 5, 1962, 394–397.

Dechter, R. (1991). Constraint networks. *Encyclopedia of Artificial Intelligence* John Wiley, New York (1991) 276-285.

Freuder, E. and Mackworth, A., eds. (1994). *Constraint-based reasoning*. MIT Press, Cambridge, MA.

Gent, Ian P. and Walsh, Toby (1994). Easy problems are sometimes hard. *Artificial Intelligence*, (70)1-2, 335–345.

Ginsberg, M. (1993). Dynamic Backtracking. *Journal of Artificial Intelligence*, Vol. 1, 25–46.

Gomes, C.P. and Selman, B. (1997). Problem structure in the presence of perturbations. *Proc. AAAI-97*, New Providence, RI, 221–226.

Gomes, C.P., Selman, B.,and Crato, N. (1998a). Heavy-Tailed Phenomena in Combinatorial Search, 1998. (submitted for publication)

Gomes, C.P. and Selman, B., McAloon, K., and Tretkoff C. (1998b). Randomization in Backtrack Search: Exploiting Heavy-Tailed Profiles for Solving Hard Scheduling Problems. To appear in: *Proc. AIPS-98*.

Kamath, A.P., Karmarkar, N.K., Ramakrishnan, K.G., and Resende, M.G.C. (1993). An Interior Point Approach to Boolean Vector Function Synthesis. *Proc. 36th MSCAS*, 185–189.

Kautz, H. and Selman, B. (1996). Pushing the envelope: planning, propositional logic, and stochastic search. *Proc. AAAI-1996*, Portland, OR.

Li, Chu Min and Anbulagan (1997). Heuristics based on unit propagation for satisfiability problems. *Proc. IJCAI-97*, Kyoto, Japan, 1997.

Luby, M., Sinclair A., and Zuckerman, D. (1993). Optimal speedup of Las Vegas algorithms. *Information Process. Lett.*, 17, 1993, 173–180.

Mandelbrot, Benoit, B. (1960). The Pareto-Lévy law and the distribution of income. *International Economic Review* 1, 79–106.

McAloon, K., Regin, J-C., Tretkoff C. and Wetzel G. (1998). Constraint-Based Programming for Sports League Scheduling. Manuscript in preparation 1998.

McAloon, K., Tretkoff C. and Wetzel G. (1997). Sports League Scheduling. *Proceedings of Third Ilog International Users Meeting*, 1997.

Nemhauser, G., and Trick, M. (1997). Scheduling a major college basketball conference. Georgia Tech., Technical Report, 1997.

Oddi A. and Smith, S. (1997) Stochastic procedures for generating feasible schedules. *Proc. AAAI-97*, New Providence, RI, 1997.

Puget, J-F., and Leconte, M. (1995). Beyond the Black Box: Constraints as objects. *Proceedings of ILPS'95*, MIT Press, 513–527.

Samorodnitsky, Gennady and Taqqu, Murad S. (1994). *Stable Non-Gaussian Random Processes: Stochastic Models with Infinite Variance*, Chapman and Hall, New York.

Schreuder, J. A. M. (1992). Combinatorial Aspects of Construction of Competition Dutch Professional Football Leagues, *Discrete Applied Mathematics* 35 (1992) 301-312.

Selman, B. and Kautz, H. (1993). Domain-independent extensions to GSAT: solving large structured satisfiability problems. *Proc. IJCAI-93*, Chambéry, France, 290–295.

Smith, B. and Grant S.A. (1996). Sparse constraint graphs and exceptionally hard problems. *IJCAI-95*, 646–651, 1995. Full version in AIJ (Hogg et al. 1996).

Wetzel, G. and Zabatta, F. (1998). CUNY Graduate Center CS Technical Report, 1998.

Algorithm Portfolios

Carla Gomes and Bart Selman
Dept. of Computer Science
Cornell University
Ithaca, NY 14850
{gomes,selman}@cs.cornell.edu

Abstract

Stochastic algorithms are among the best methods for solving computationally hard search and reasoning problems. The runtime of such procedures can vary significantly from instance to instance and, when using different random seeds, on the same instance. One can take advantage of such differences by combining several algorithms into a portfolio, and running them in parallel or interleaving them on a single processor. We provide a detailed evaluation of the portfolio approach on distributions of hard combinatorial search problems. We show under what conditions the portfolio approach can have a dramatic computational advantage over the best traditional methods.

1 Introduction

Randomized algorithms are among the best current algorithms for solving computationally hard problem. Most local search methods for solving combinatorial optimization problems have a stochastic component, both to generate an initial candidate solution, as well as to choose among good local improvements during the search. Complete backtrack-style search methods often also use an element of randomness in their value and variable selection in case of ties. The runtime of such algorithms varies from run to run on the same problem instance, and therefore can be characterized by a probability distribution. The performance of algorithms can also vary dramatically among different problem instances. In this case, we want to consider the performance profile of the algorithms over a spectrum of problem instances.

Given the diversity in performance profiles among algorithms, various approaches have been developed to optimize the overall performance of algorithms, taking into account

computational resource constraints. These considerations led to the development of anytime algorithms (Dean and Boddy 1988), decision theoretic metareasoning and related approaches (Horvitz and Zilberstein 1996; Russell and Norvig 1995).

Despite the numerous results obtained in these areas, so far they have not been exploited much by the traditional communities that study hard computational problems, such as operations research (OR), constraint satisfaction (CSP), theorem proving, and the experimental algorithms community. In order to bridge this gap, we study the possibility of combining algorithms into portfolios in the context of the recent results concerning the inherent complexity of computationally hard search and reasoning problems (Gomes and Selman 97; Huberman *et al.* 1997). We analyze the performance profiles of several of the state-of-the-art search methods on a distribution of hard search problems encoded as Constraint Satisfaction problems (CSP), and as Mixed Integer Programming problems (MIP). Our study reveals several interesting problem classes where a portfolio approach gives a dramatic improvement in terms of overall performance, compared to a single algorithm approach. In addition, we also show that a good strategy for designing a portfolio is to combine many short runs of the *same* algorithm. The effectiveness of such portfolios explains the common practice of "restarts" for stochastic procedures, where the same algorithm is run repeatedly with different initial seeds for the random number generator.

Our analysis also provides some new directions for designing good heuristic algorithmic strategies. For example, in our MIP domain, we find that a depth-first search strategy is preferable over the more standard best-bound strategy. On a single processor, the best-bound strategy is more robust than a depth-first strategy. That is, both the expected runtime and variance of best-bound approaches tend to be smaller than the corresponding values for depth-first strategies. However, when one has several processors available, *e.g.*, in a compute cluster, a collection of depth-first runs outperforms a set of best-bound runs. The key is that depth-first is in a sense a more "audacious" strategy. It has a much larger variance than the best-bound strategy and has a non-negligable chance of finding solutions on very short runs. By running an ensemble of such "risky" strategy, one can outperform the more conservative best-bound strategy. In fact, one obtains a smaller expected overall runtime and a smaller variance when using a portfolio of depth-first runs. These insights suggest that when multiple compute resources are available, a design aiming for high-variance search methods, especially with a non-neglible probability of finding solutions with short runs, may result in the best overall strategy.

Overall, our results suggest that the various ideas on flexible computation can indeed play a significant role in algorithm design, complementing the more traditional methods for solving computationally hard search and reasoning problems. We hope that this work will push these ideas closer to practical applications.

2 Search procedures and problem domains

2.1 Randomization of Backtrack Search

We consider a general technique for adding randomization to complete, systematic, back-track search procedures. Such procedures construct a solution incrementally. At each step a heuristic is used to select an operation to be applied to a partial solution, such as assigning a value to an unassigned variable. Eventually either a complete solution is found, or the algorithm determines that the current partial solution is inconsistent. In the latter case, the algorithm backtracks to an earlier point in its search tree.

If several choices are heuristically determined to be equally good, then a deterministic algorithm applies some fixed rule to pick one of the operations, for example, by selecting the variables in lexicographic order. The most obvious place to apply randomization, therefore, is in this tie-breaking step: if several choices are ranked equally, choose among them at random. Even this simple modification can dramatically change the behavior of a search algorithm, as we will see below. However, if the heuristic function is particular powerful, it may rarely assign more than one choice the highest score. To handle this situation, we can introduce a "heuristic equivalence" parameter to the algorithm. Setting the parameter to a value H greater than zero means all choices that receive scores within H-percent of the highest score are considered equally good. This expands the choice set for random tie-breaking.

With these changes, each run of the backtrack search algorithm on a particular instance will differ in the order in which choices are made and potentially in time to solution. We should note that introducing randomness in the branching variable selection does not affect the completeness of the backtrack search. Some basic bookkeeping ensures that the procedures do not revisit any previously explored part of the search space, which means that we can still determine inconsistencies, unlike local search methods. The bookkeeping mechanism involves some additional information, where for each variable on the stack, we keep track of which assignments have been tried so far.

We randomized the Ilog constraint solver engine for our experiments on constraint satisfaction formulations. Ilog provides a powerful C++ constraint programming library (Puget and Leconte 1995). We randomized the first-fail heuristic and various variants of the Brelaz selection rule, which has been shown to be effective on graph coloring style problem domains (Brelaz 1979).

2.2 Randomization of Branch-and-Bound

The standard approach used by the OR community to solve mixed integer programming problems (MIP) is branch-and-bound search. First, a linear program (LP) relaxation of the problem instance is considered. In such a relaxation, all variables of the problem are treated as continuous variables. If the solution to the LP relaxation problem has non-integer values

for some of the integer variables, we have to branch on one of those variables. This way we create two new subproblems (nodes of the search tree), one with the floor of the fractional value and one with the ceiling. (For the case of binary (0/1) variables, we create an instance with the variable set to 0 and another with the variable set to 1.) The standard heuristic for deciding which variable to branch on is based on the degree of infeasibility of variables ("max infeasibility variable selection"). That is, we select the variable whose non-integer part in the solution of the LP relaxation is closest to 0.5. Informally, we pick the variable whose value is "least integer".

Following the strategy of repeatedly fixing integer variables to integer values will lead at some point to a subproblem with an overall integer solution (provided we are dealing with a feasible problem instance). (Note we call any solution where all the integer variables have integer values an "integer solution".) In practice, it often happens that the solution of the LP relaxation of a subproblem already is an integer solution, in which case we do not have to branch further from this node.

Once we have found an integer solution, its objective function value can be used to prune other nodes in the tree, whose relaxations have worse values. This is because the LP relaxation bounds the optimal solution of the problem. For example, for a minimization problem, the LP relaxation of a node provides a lower-bound on the best possible integer solution.

A critical issue that determines the performance of branch-and-bound is the way in which the next node to expand is selected. The standard approach, in OR, is to use a best-bound selection strategy. That is, from the list of nodes (subproblems) to be considered, we select the one with the best LP bound. (This approach is analogous to an A* style search. The LP relaxation provides an admissible search heuristic.)

The best-bound node selection strategy is particularly well-suited for reaching an optimal solution (because of the greedy guidance), which has been the traditional focus of much of the research in OR. One significant drawback of this approach is that it may take a long time before the procedure finds an integer solution, because of the breadth first flavor of the search. Also, the approach has serious memory requirements because the full fringe of the tree has to be stored.

Given problems that have a difficult feasibility part, the best-bound approach may take too long before reaching an integer solution. (Note that an integer solution is required before any nodes can be pruned.) In our experiments, we therefore also considered a depth-first node selection strategy. Such a strategy often quickly reaches an integer solution, but may take longer to produce an overall optimal value.

In our experiments, we used a state-of-the-art MIP programming package, called CPLEX. CPLEX provides a set of libraries that allows one to customize the branch-and-bound search strategy. For example, one can vary node selection, variable selection, variable setting strategies, the LP solver, etc. We used the default settings for the LP solver, which is for the first node primal-simplex and for subsequent nodes dual-simplex. We modified

the search strategies to include some level of randomization. We randomized the variable selection strategy by introducing noise in the ranking of the variables, based on maximum infeasibility. Note that the completeness of the search method is maintained. We have experimented with several other randomization strategies. For example, in CPLEX one can assign an apriori variable ranking, which is fixed throughout branch-and-bound. We experimented by randomizing this apriori ranking. We found, however, that the dynamic randomized variable selection strategy, as described above, is more effective.

2.3 Problem Domains

In order to study the performance profile of different search strategies, we derive generic distributions of hard combinatorial search problems from the domain of finite algebra. In particular, we consider the quasigroup domain. A quasigroup is an ordered pair $(Q, \cdot)$, where Q is a set and $(\cdot)$ is a binary operation on Q such that the equations $a \cdot x = b$ and $y \cdot a = b$ are uniquely solvable for every pair of elements a, b in Q. The *order N* of the quasigroup is the cardinality of the set Q. The best way to understand the structure of a quasigroup is to consider its N by N multiplication table as defined by its binary operation. The constraints on a quasigroup are such that its multiplication table defines a *Latin square*. This means that in each row of the table, each element of the set Q occurs exactly once; similarly, in each column, each element occurs exactly once (Denes and Keedwell 1974).

An *incomplete* or *partial latin square P* is a partially filled N by N table such that no symbol occurs twice in a row or a column. The *Quasigroup Completion Problem* is the problem of determining whether the remaining entries of the table can be filled in such a way that we obtain a complete latin square, that is, a full multiplication table of a quasigroup. We view the pre-assigned values of the latin square as a *perturbation* to the original problem of finding an arbitrary latin square. Another way to look at these pre-assigned values is as a set of additional problem constraints to the basic structure of the quasigroup.

There is a natural formulation of the problem as a Constraint Satisfaction Problem. We have a variable for each of the N^2 entries in the multiplication table of the quasigroup, and we use constraints to capture the requirement of having no repeated values in any row or column. All variables have the same domain, namely the set of elements Q of the quasigroup. Pre-assigned values are captured by fixing the value of some of the variables.

Colbourn (1983) showed the quasigroup completion problem to be NP-complete. In previous work, we identified a clear phase transition phenomenon for the quasigroup completion problem (Gomes and Selman 1997). See Figure 1. From the figures, we observe that the costs peak roughly around the same ratio (approximately 42% pre-assignment) for different values of N. (Each data point is generated using 1,000 problem instances. The pre-assigned values were randomly generated.) This phase transition with the corre-

sponding cost profile allows us to tune the difficulty of our problem class by varying the percentage of pre-assigned values.

An interesting application area of latin squares is the design of statistical experiments. The purpose of latin squares is to eliminate the effect of certain systematic dependency among the data (Denes and Keedwell 1974). Another interesting application is in scheduling and timetabling. For example, latin squares are useful in determining intricate schedules involving pairwise meetings among the members of a group (Anderson 1985). The natural perturbation of this problem is the problem of completing a schedule given a set of pre-assigned meetings.

The quasigroup domain has also been extensively used in the area of automated theorem proving. In this community, the main interest in this domain has been driven by questions regarding the existence and nonexistence of quasigroups with additional mathematical properties (Fujita et al. 1993; Lam et al. 1989).
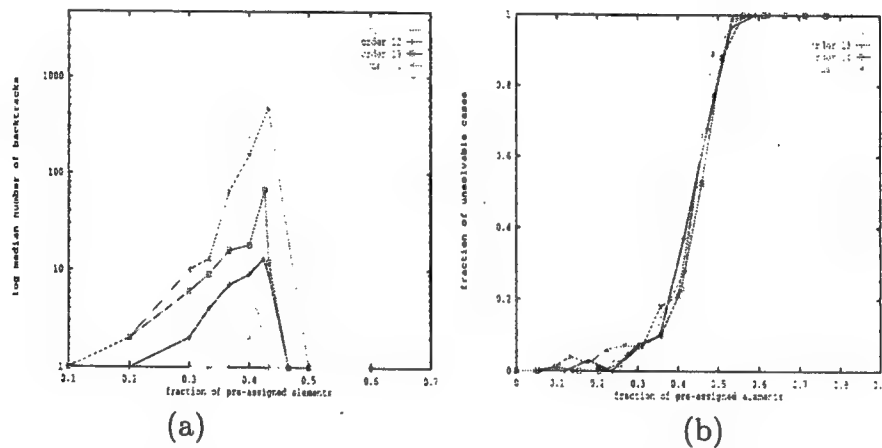

Figure 1: (a) Cost profile, and (b) phase transition for the quasigroup completion problem (up to order 15).

In our experiments we were also interested in problems that combine a hard combinatorial component with numerical information. Integrating numerical information into standard AI formalism is becoming of increasing importance. For example, in planning, one would like to incorporate resource constraints or a measure of overall plan quality. We considered examples based on logistics planning problems, formulated as mixed integer programming problems. These formulations extend the traditional AI planning approach by combining the hard constraints of the planning operators, initial state, and goal state, with a series of soft contraints capturing resource utilization. For example, one can require that trucks are loaded as close as possible to their maximum capacity. Such formulations have been shown to be very promising for modeling AI planning problems (Kautz and

Walser 1999; Vossen *et al.* 1999). Experimentation with the CPLEX MIP solver showed that these problem instances are characterized by a non-trivial feasibility component.[1]
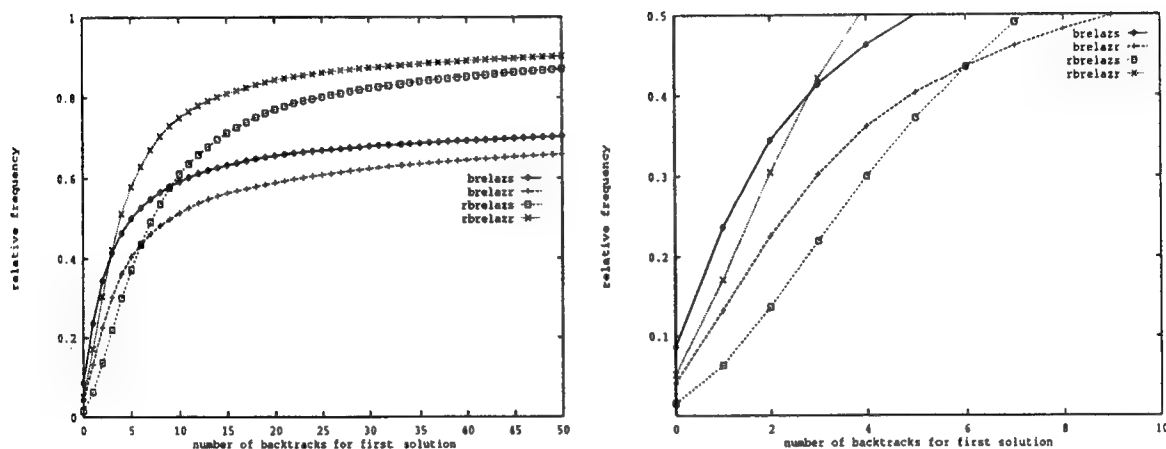
3 Computational Cost Profiles

Figure 2: Cost profiles for the quasigroup completion problem for a range of search heuristics.

We start by considering the computational cost of solving the quasigroup completion problem for different search strategies. As our basic search procedure, we use a complete backtrack-style search method. The performance of such procedures can vary dramatically depending on the way one selects the next variable to branch on (the "variable selection strategy") and in what order the possible values are assigned to a variable (the "value selection strategy"). There is a large body of work in both the CSP and OR communities exploring different search strategies.

One of the most effective strategies is the so-called First-Fail heuristic.[2] In the First-Fail heuristic, the next variable to branch on is the one with the smallest remaining domain (*i.e.*, in choosing a value for the variable during the backtrack search, the search procedure has the fewest possible options left to explore — leading to the smallest branching factor). We consider a popular extension of the First-Fail heuristic, called the Brelaz heuristics (Brelaz 1979). The Brelaz heuristic was originally introduced for graph coloring procedures. It

---

[1] We thank Henry Kautz and Joachim Walser for providing us with MIP formulations of the logistic planning problems.

[2] It's really a prerequisit for any reasonable bactrack-style search method. In theorem proving and Boolean satisfiability, the rule corresponds to the powerful unit-propagation heuristic.

provides one of the most powerful graph-coloring and general CSP heuristics (Trick and Johnson 1996).

The Brelaz heuristic specifies a way for breaking ties in the First-fail rule: If two variables have equally small remaining domains, the Brelaz heuristic proposes to select the variable that shares constraints with the largest number of the remaining unassigned variables. A natural variation on this tie-breaking rule is what we call the "reverse Brelaz" heuristic, in which preference is given to the variable that shares constraints with the *smallest* number of unassigned variables. Any remaining ties after the (reverse) Brelaz rule are resolved randomly. One final issue left to specify in our search procedure is the order in which the values are assigned to a variable. In the standard Brelaz, value assignment is done in lexicographical order (*i.e.*, systematic). In our experiments, we consider four stragies:

- *Brelaz-S* — Brelaz with systematic value selection,

- *Brelaz-R* — Brelaz with random value selection,

- *R-Brelaz-S* — Reverse Brelaz with systematic value selection, and

- *R-Brelaz-R* — Reverse Brelaz with random value selection.

Figure 2 shows the performance profile of our four strategies on an instance of the quasigroup completion problem (order 20, 10% preassigned). Each curve gives the cumulative distribution obtained for each strategy by solving the problem 10,000 times. The cost (horizontal axis) is measured in number of backtracks, which is directly proportional to the total runtime of our strategies. For example, the figure shows that R-Brelaz-R, finished roughly 80% of the 10,000 runs in 15 backtracks or less. The left panel of the figure shows the overall profile; the right panel gives the initial part of the profile.

Note that that R-Brelaz-R dominates R-Brelaz-S over the full profile. In other words, the cumulative relative frequency curve for R-Brelaz-R lies above that of R-Brelaz-S at every point along the x-axis. Brelaz-S, in turn, strictly dominates Brelaz-R.

From the perspective of combining algorithms, what is most interesting, however, is that in the initial part of the profile (see right panel of Figure 2), Brelaz-S dominates R-Brelaz-R. Intuitively, Brelaz-S is better than R-Brelaz-R at finding solutions *quickly*. However, in the latter part of the cumulative distribution, R-Brelaz-R dominates Brelaz-S. In a sense, R-Brelaz-R gets relatively better when the search gets harder. As we will see in the next section, we can exploit this in our algorithm portfolio design.

In Figure 3, we compare the runtime profile of a depth-first strategy with a best-bound strategy to solve a hard feasibility problem in the logistics domain, formulated as a mixed integer programming problem. The search is terminated when an optimal or near-optimal (<10% from optimal) solution is found, but without the requirement of proving
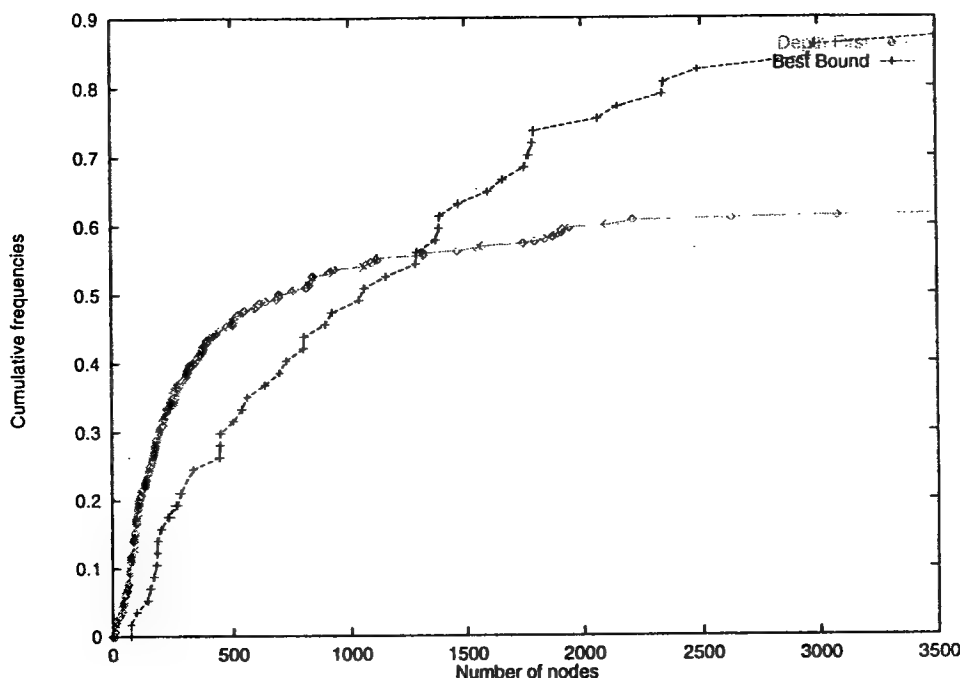
Figure 3: Cost profiles for a logistics planning problems for depth-first and best-bound search strategies.

optimality.[3] The figure shows the cumulative distribution of solution time (in number of expanded nodes). For example, with 500 or less nodes, the depth-first search finds a solution on approximately 50% of the runs. Each run had a time limit of 5000 seconds. As we see from the figure, the depth-first search initially outperforms the best-bound search. However, after more than 1500 node expansions, the best-bound becomes more effective. For example, best-bound finds a solution on approximately 75% of the runs with 2000 node expansions or less. In contrast, depth-first search can only find solutions on 55% of the runs with the same number of node expansions. This data is consistent with the observation above that best-bound may take some time to find an intial integer solution. However, once such an initial integer solution is found, best-bound becomes more effective.

We now consider the runtime distributions more closely. Figure 4 gives a log-log plot of the complement of the cumulative distribution for the depth-first procedure. For example, from this plot, we see that after 10,000 nodes, approximately 30% of the runs have not yet found the solution. The figure shows a near linear behavior over several orders of magnitude. This is an indication of so-called heavy-tailed behavior which often characterizes

---

[3]One should be careful to distinguish between finding an optimal integer solution and proving that this is indeed *the* optimal solution. Our interest lies in problems where the proof of optimality can be beyond reach of any procedure; however, we can often still find good quality solution.
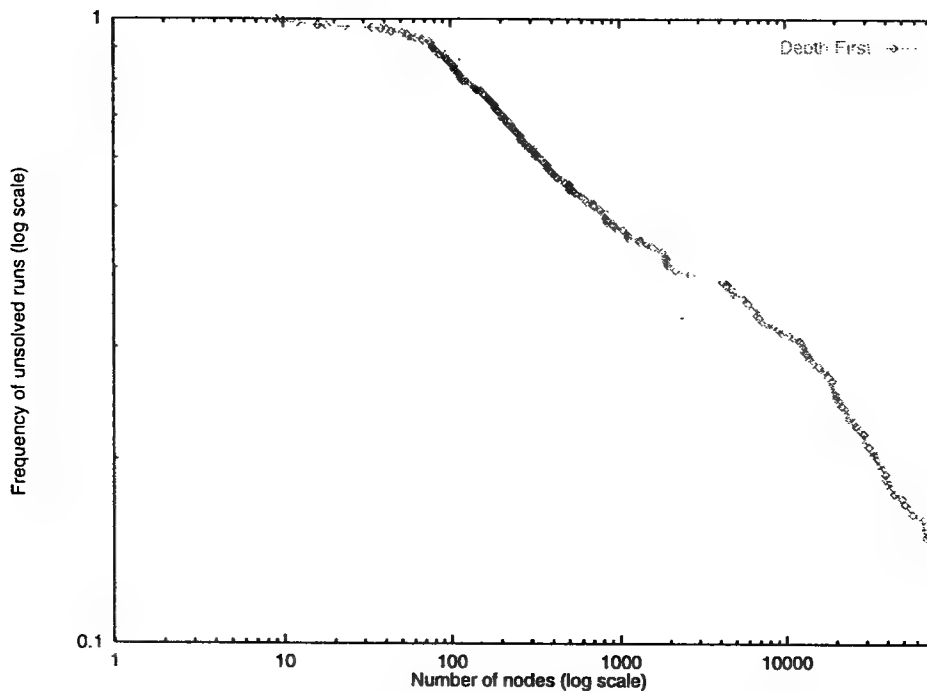
Figure 4: Heavy-tailed behavior of depth-first search.

complete search methods (Gomes *et al.* 1997). In a sense, the time till solution behaves in a very erratic manner: very long runs occur much more frequently than one might expect. Best-bound also appears to exhibit heavy-tailed behavior, but less dramaticly than that for depth-first search.

This erratic search behavior is related to the feasibility part of the search. Even though the main focus of our study was the study of the runtime distributions for feasibility, preliminary results on the characterization of the runtime distributions for proving optimality indicate that such distributions do not appear to be heavy-tailed. We conjecture that indeed proving optimality does not produce heavy-tails since the entire search space has to be explored. This is consistent with the results by Frost *et al.* (1997) on constraint satisfaction problems. They show that standard (not heavy-tailed) distributions, such as the Weibull and log-normal distribution, underly the cost of proving inconsistency. Further experimentation is required.

In the next section we show how the large variance in search methods, as characterized by heavy-tailed behavior, can be exploited by combining algorithms into portfolios or running multiple copies of the same algorithm.

4 Portfolio Design

A *portfolio of algorithms* is a collection of different algorithms and/or different copies of the same algorithm running on different processors (Gomes and Selman 97; Huberman *et al.* 1997). Here we consider the case of independent runs without interprocess communication.[4]

We are considering Las Vegas type algorithms, *i.e.*, stochastic algorithms that always return a model satisfying the constraints of the search problem or demonstrate that no such model exists (Motwani and Raghavan 1995). The computational cost of the portfolio is therefore a random variable. The *expected* computational cost of the portfolio is simply the expected value of the random variable associated with the portfolio and its *standard deviation* is a measure of the "dispersion" of the computational cost obtained when using the portfolio of algorithms. In this sense, the standard deviation is a measure of the risk inherent to the portfolio.

The main motivation to combine different algorithms into a portfolio is to improve on the performance of the component algorithms, mainly in terms of expected computational cost but also in terms of the overall *risk*. As we will show, some portfolios are strictly preferrable to others, in the sense that they provide a lower risk and also a lower expected computational cost. However, in some cases, we cannot identify any portfolio within a set that is the best, both in terms of expected value and risk. This set of portfolios corresponds to the *efficient set* or *efficient frontier*, following terminology used in the theory of mathematical finance. Within this set, in order to minimize the risk, one has to deteriorate the expected value or, in order to improve the expected value of the portfolio, one has to increase the risk.

In this context, where we characterize a portfolio in terms of its mean and variance, combining different algorithms into a portfolio only makes sense if they exhibit different probability profiles and none of them *dominates* the others over the whole spectrum of problem instances. An algorithm A dominates algorithm B if the cumulative frequency distribution of algorithm A lies above the cumulative frequency distribution of algorithm B for all points.[5]

Let us consider a set of two algorithms, *algorithm* 1 and *algorithm* 2. Let us associate a random variable with each algorithm: A1 — the number of backtracks that algorithm 1 takes to find the first solution or to prove that a solution does not exist; A2 — the number of backtracks that algorithm 2 takes to find the first solution or to prove that a solution does not exist.

Let us assume that we have N processors and that we design a portfolio using $n1$ processors with *algorithm* 1 and $n2$ processors with *algorithm* 2. So, $N = n1 + n2$. Let us define the random variable associated with this portfolio: X — the number of backtracks

---

[4]One can also consider the somewhat more general case of interleaving the execution of algorithms on one or more processors.

[5]Another criterion for combining algorithms into a portfolio is given by the algorithm *covariance*.

that the portfolio takes to find the first solution or to prove that a solution does not exist.

The probability distribution of X is a "weighted" probability distribution of the probability distributions of *algorithm* 1 and *algorithm* 2. More precisely, the probability that $X = x$ is given by the probability that one processor takes exactly x backtracks and all the other ones take x or more backtracks to find a solution or to prove that a solution does not exist.

Let us assume that we have N processors and our portfolio consists of N copies of *algorithm* 1. In this case, P[X=x] is given by the probability that one processor take exactly x backtracks and the other $N-1$ take more than x backtracks, plus the probability that two processors take exactly x backtracks and the other (N-2) one takes more than x backtracks, etc., plus the probability that all the processors take exactly x backtracks to find a solution or to prove that a solution does not exist. The following expression gives the probability function for such a portfolio.

Given N processors, and let $n1 = N$ and $n2 = 0$. P[X=x] is given by

$$\sum_{i=1}^{N} \binom{N}{i} P[A1 = x]^{i} P[A1 > x]^{(N-i)}$$

To consider two algorithms, we have to generalize the above expression, considering that $X = x$ can occur just within the processors that use *algorithm* 1, or just within the processors that use *algorithm* 2 or within both. As a result, the probability function for a portfolio with two algorithms, is given by the following expression:

Given N processors, $n1$ such that $0 <= n1 <= N$, and $n2 = N - n1$, P[X=x] is given by

$$\sum_{i=1}^{N} \sum_{i'=0}^{n1} \binom{n1}{i'} P[A1 = x]^{i'} P[A1 > x]^{(n1-i')} \times$$

$$\binom{n2}{i''} P[A2 = x]^{i''} P[A2 > x]^{(n2-i'')}]$$

The value of i'' is given by $i'' = i - i'$, and the term in the summation is 0 whenever $i'' < 0$ or $i'' > n2$.

In the case of a portfolio involving two algorithms the probability distribution of the portfolio is a summation of a product of two expressions, each one corresponding to one algorithm. In the case of a portfolio comprising M different algorithms, this probability function can be easily generalized, by having a summation of a product of M expressions, each corresponding to an algorithm.

Once we derive the probability distribution for the random variable associated with the portfolio, the calculation of the its *expected value* and *standard deviation* is straightforward.

5 Empirical Results for Portfolio Design

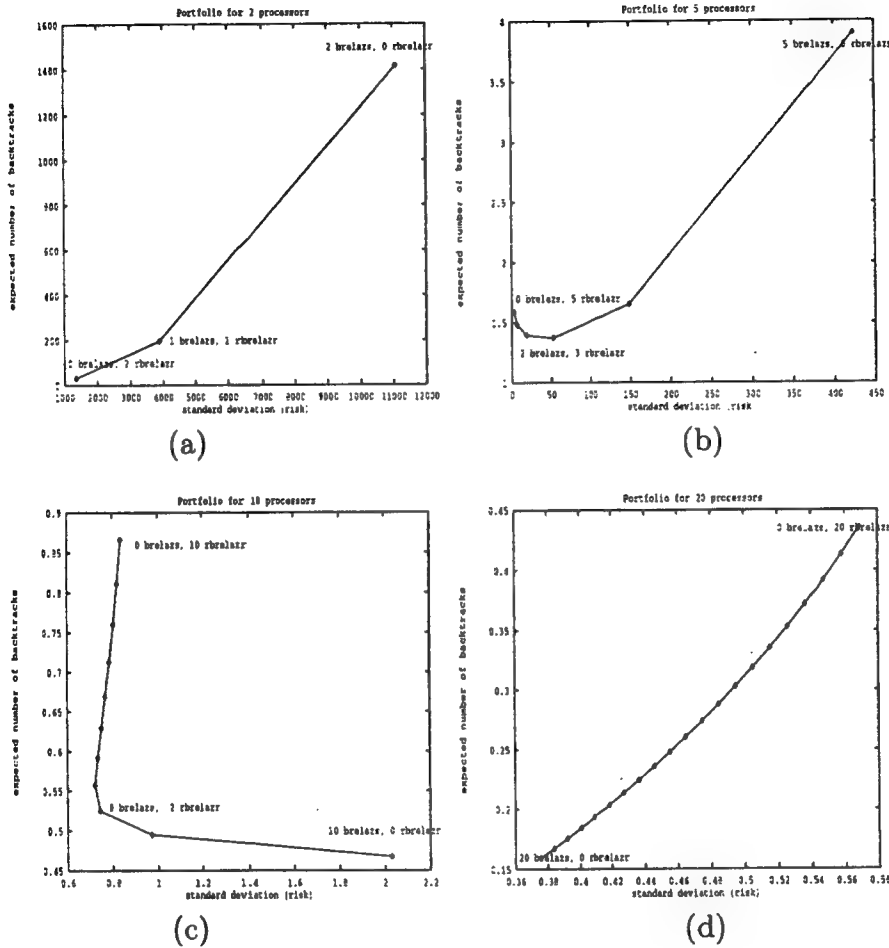## 5.1 Constraint Satisfaction

Figure 5: Portfolio combining Brelaz and R-Brelaz-R for solving a quasigroup completion instance using two (a), five (b), ten (c), and twenty processors (d).

We now derive different portfolios based on the runtime profiles given in Figure 2 (Section 3). Note that this is an interesting case from the portfolio design perspective because Brelaz-S dominates in the initial part of the distribution, whereas R-Brelaz-R dominates in the latter part.

Figure 5 gives the expected runtime values and the standard deviations of portfolios for 2, 5, 10, and 20 processors. (Results derived using the formula given above.) We see that for 2 processors (Figure 5(a)), the portfolio consisting of two copies of R-Brelaz-R has

51

the lowest expected runtime *and* the lowest standard deviation. This portfolio dominates the two other 2-processor portfolios.

When we increase the number of processors, we observe an interesting shift in the optimal portfolio mix. For example, for 5 processors, using 2 copies of Brelaz-S gives a better expected value at only a slight increase in the risk (standard deviation), compared to zero Brelaz-S. In the case of five processors, the efficient set comprises four portfolios: one with 5 R-Brelaz-R, one with 1 Brelaz-S and 4 R-Brelaz-R, one with 2 Brelaz-S and 3 R-Brelaz-R, and one with 3 Brelaz-S and 2 R-Brelaz-R. There is no clear dominant portfolio among those three. In this set, one has to trade a decrease in expected runtime for an increase in variance: in order to minimize the expected runtime, the best portfolio is 3 Brelaz-S and 2 R-Brelaz-R; in order to minimize the risk (variance) the best portfolio corresponds to 5 R-Brelaz-R.

The situation changes even more dramatically if we increase the number of processors. In particular, with 20 processors (Figure 5d), the best portfolio corresponds to using copies of the Brelaz-S strategy on all processors, obtaining the lowest expected value and the lowest standard deviation. The intuitive explanation for this is that by running many copies of Brelaz-S, we have a good chance that at least one of them will find a solution quickly. This result is consistent with the common use of "random restarts" in stochastic search methods in practical applications. Our portfolio analysis also gives the somewhat counter-intuitive result that, even when given two stochastic algorithms, where neither strictly dominates the other, running multiple copies of a single algorithm is preferrable to a mix of algorithms.

5.2 Mixed Integer Programming

In section 3 we have shown that there are several interesting trade-offs between depth-first branch-and-bound versus best-bound branch-and-bound. In particular, depth-first search performs better early on in the search, whereas best-bound is better on longer runs. Again, as with constraint satisfaction methods in the quasigroup domain, we will show how a portfolio approach can be used to effectively combine the best features of each search strategy.

In figure 6, we consider a range of portfolios for solving our feasibility problem for the logistics domain, considering situations from one processor to twenty processors (the same instance as the one considered in figure 3). The plot gives the expected runtime and standard deviation for different ways of combining a branch-and-bound search procedure using depth-first search and best-bound search. From this plot we see that the best choice, when using a single processor, in terms of minimizing expected runtime and standard deviation, corresponds to running branch-and-bound with best-bound. Best-bound remains the best strategy when we consider two processors, *i.e.*, the best portfolio consists of running branch-and-bound with best-bound on both processors. The expected runtime of such a
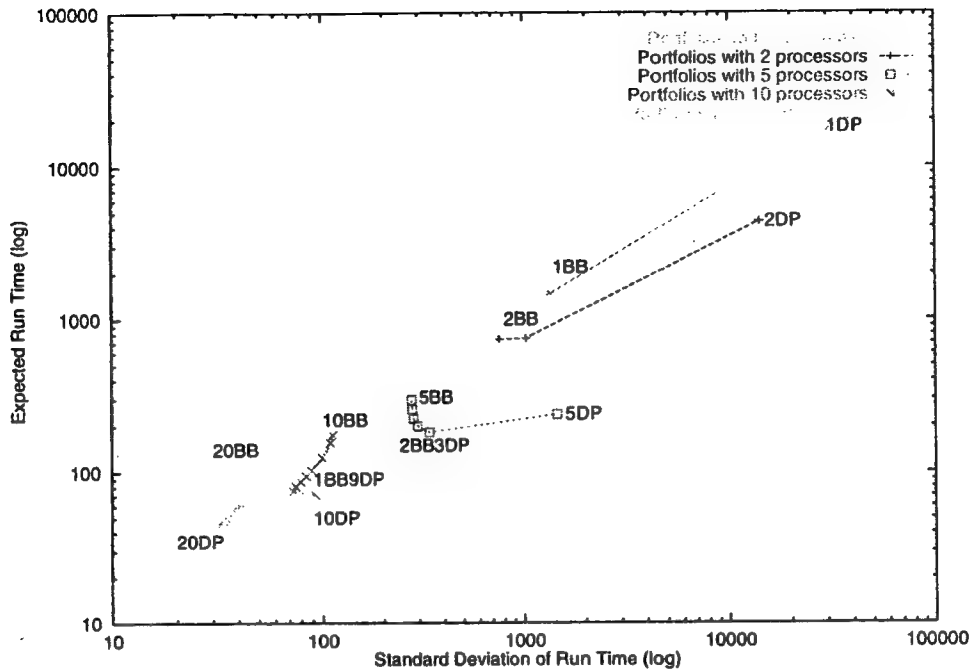
Figure 6: A range of portfolios for the MIP formulation of logistics planning (expected run time).

strategy is approximately 700 nodes with a standard deviation of around 750. Contrast these values, for example, with the much higher values corresponding to the strategy of running branch-and-bound with depth-first on both processors (average 4397 and standard deviation of 14112).

The mixing strategy changes as we increase the number of processors or the amount of interleaving. For example, for the case of ten processors, the best strategies are 9DF/1BB and 10DF/0BB. (We use the notation xDF/yBB to mean running x depth-first processes and y best-bound processes.) These strategies give both a low expected runtime and a low standard deviation. There is no clear dominant strategy among those two (efficient set). In this set, one has to trade a decrease in expected runtime for an increase in variance: in order to minimize the expected runtime, the best portfolio is 10DF/0BB; in order to minimize the risk (variance) the best portfolio corresponds to 9DF/1BB. It is interesting to observe that, in the case of 20 processors, the best strategy corresponds to only using depth-first search (i.e., 20DF/0BB).

Figure 7 shows the total computational cost for the different portfolios. In other words, rather than just considering the time for solution as in figure 6, figure 7 factors in the number of processors.

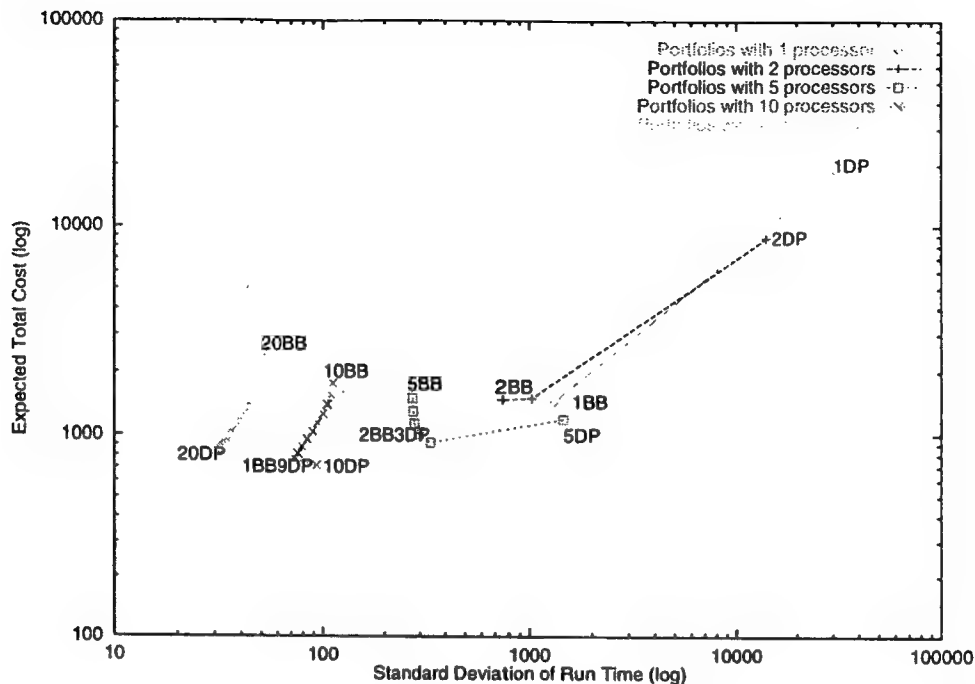Figure 8, panel (a), shows that the reduction in expected runtime decreases at a very

53

Figure 7: A range of portfolios for the MIP formulation of logistics planning (expected total cost).

slow rate, for more than 10 processors. In fact, figure 8, panel (b), shows that, the most cost-effective solution[6] is obtained for a 10 processor portfolio.

6   Restart Strategies

A key intuition behind the effectiveness of the portfolio approach is that it takes advantage of relatively short runs. Having more processors available increases the probability of encountering a relatively short run. One can simulate a multi-processor approach by running several copies of the search methods interleaved on a single processor. A related strategy is to use a restart strategy, where one periodically restarts the randomized search with a new random seed. As discussed in section 3, backtrack search cost profiles are often heavy-tailed, with runtimes varying over several orders of magnitude on different runs of the same instance. Restart strategies reduce the extreme variance that is inherent in backtrack search procedures. In this section we formalize the restart strategy of a complete randomized backtrack search and we present results for the restart strategy applied to depth-first and best-bound. For related work on restarts, see *e.g.*, Aldous and Vazirani 1994; Alt *et al.* 1996; Ertel 1991; Luby *et al.* 1993; and Selman and Kirkpatrick 1996.)

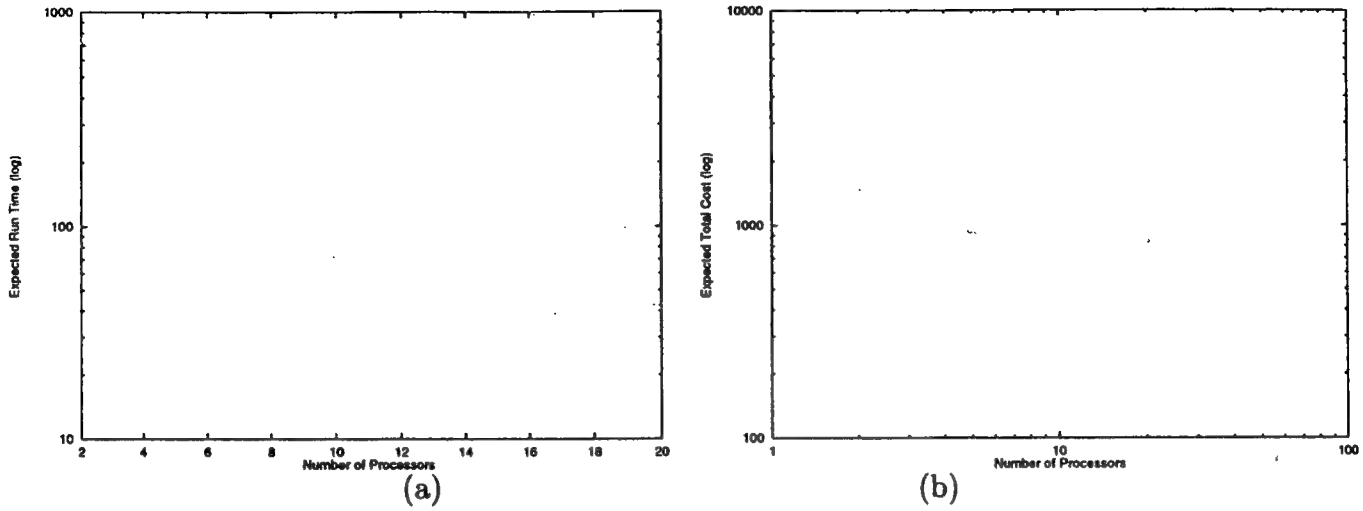[6]We consider the total cost, *i.e*, the time to find a solution times the number of processors

Figure 8: Expected time (a) and expected total cost (b) of optimal portfolio for different numbers of processors.

6.1 A Formal Characterization of Restarts

Given a randomized backtrack search procedure, let us consider the number of choice points (or backtracks) performed by such a procedure. We introduce random variable B, such that,

> B is the number of choice points that the backtrack search procedure takes to find a solution or prove that it does not exist. $B = \{1, 2, \cdots\}$

Now consider a Rapid Randomized Restarts (RRR) strategy for running our backtrack procedure: run the procedure up to a fixed number of choice points c (the cutoff); if the procedure finds a solution or proves that no solution exists, then RRR has also found a solution (or proven that no solution exists) and stops; otherwise restart the backtrack procedure from the beginning (using an independent random seed) for another c decision events, and so on. We associate with RRR the random variable S, such that,

> S is the number of choice points that RRR takes to find a solution or prove that no solution exists. $S = \{1, 2, \cdots\}$

Let's define a "run", as the execution of the randomized backtrack search method for up to c steps. We now define the random variable R, such that,
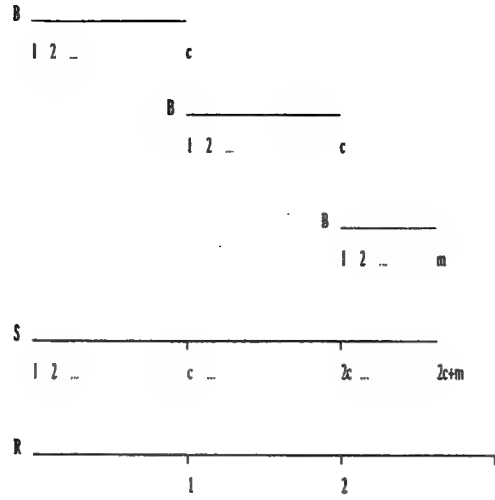
> R is the number of runs executed by RRR.

Figure 9: B — number of choice points searched by the randomized backtrack procedure (running with cutoff c); S — number of choice points searched by RRR; R — number of runs executed by RRR. In this case, a solution was found in the third run ($R = 3$), with a total of $2c + m$ choice points ($S = 2c + m$). The third run of the randomized backtrack search method took $m \leq c$ choice points.

Figure 9 illustrates how the different random variables relate to each other. The runs executed by RRR are independent (no information is carried over between runs, and each run uses a new random seed) and therefore can be seen as a sequence of Bernoulli trials, in which the success of a trial corresponds to finding a solution (or proving that one does not exist) during a run; its probability is given by $P[B \leq c]$. Therefore, R follows a *geometric distribution* with parameter $p = P[B \leq c]$. The probability of the tail of S, $P[S > s]$, corresponds to the probability of not finding the solution in the first $\lfloor s/c \rfloor$ runs of RRR, and finding it with more than (s mod c) choice points in the next run. We obtain the following expression:

$$P[S > s] = P[B > c]^{\lfloor s/c \rfloor} \ P[B > s \bmod c] \tag{1}$$

We note that the distribution of S is not heavy-tailed since its tail exhibits exponential decay:

$$P[S > s] \leq P[B > c]^{\lfloor s/c \rfloor} = P[R > \lfloor s/c \rfloor] \tag{2}$$

56

In words, the tail of S is limited from above by the tail of R. Since R follows a geometric distribution, it has finite moments, and therefore so does S.[7]

The full distribution of S is given by the following expression:

$$P[S = s] = \begin{cases} P[B > c]^{\lfloor s/c \rfloor} \; P[B = s \bmod c] & s \bmod c \neq 0 \\ \\ P[B > c]^{\lfloor s/c \rfloor - 1} \; P[s = c] & \text{otherwise} \end{cases} \tag{4}$$

Note that the second branch of (4) corresponds to the case in which the total number of choice points executed by strategy S is a multiple of c. This situation occurs when the solution is found when the cutoff c is reached.

Based on the distribution of B, we can determine a cutoff, c, that minimizes the expected runtime of S. In our experiments, we determined the cutoff for the restart strategy (RRR) based on the empirical distribution of B, which was computed by performing $10,000$ runs of the search methods, on the same instance, with a very high cutoff.

6.2 Empirical Results for Restart Strategies

In Figure 10, we show the result of applying a strategy of fixed-length short runs ("restarts") of a randomized backtrack procedure. Figure 10(a) shows the results on a quasigroup completion instance. Without restarts and given a total of 300 backtracks, we have a failure rate of around 70%. Using restarts (every 4 backtracks), this failure rate drops to around 0.01%. The figure also shows a clear downward curve for the log-log plot of the complement-to-one of the cumulative distribution of the restart strategy, which is an indication that the heavy-tailed nature of the original cost distribution has disappeared.

Figure 11 shows the effect of different cutoff values in terms of the restart strategy on the logistics planning problem. The left panel corresponds to depth-first search, while the right panel corresponds to best-bound. Even though both panels reveal the existence of optimal cutoff values, one can observe that choosing the optimal cutoff is much more crucial for the depth-first search strategy than for the best-bound strategy. Such dramatic speed up obtained when using the optimal cutoff value with depth-first search is due to

---

[7]Heavy-tailed distributions are characterized by tails that have a *power-law* (polynomial) decay, *i.e.*, distributions which asymptotically have "heavy tails" — also called tails of the Pareto-Lévy form, *viz.*

$$P[X > x] \sim Cx^{-\alpha}, \quad x > 0 \tag{3}$$

where $0 < \alpha < 2$ and $C > 0$ are constants. Some of the moments of heavy-tailed distributions are infinite (*e.g.*, some heavy-tailed distributions have infinite mean and infinite variance, others just infinite variance, etc).
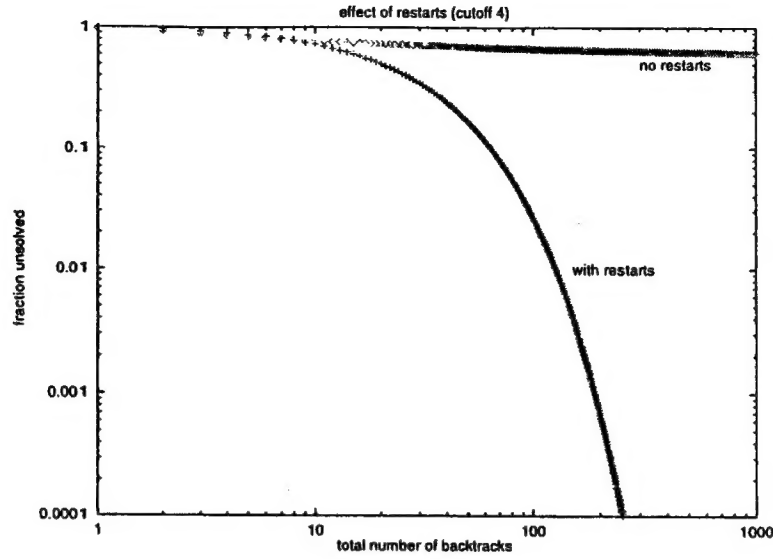
Figure 10: A rapid restart strategy to speed up backtrack search. Failure rate $(1 - F(x))$ as a function of the total number of backtracks for a quasigroup instance.
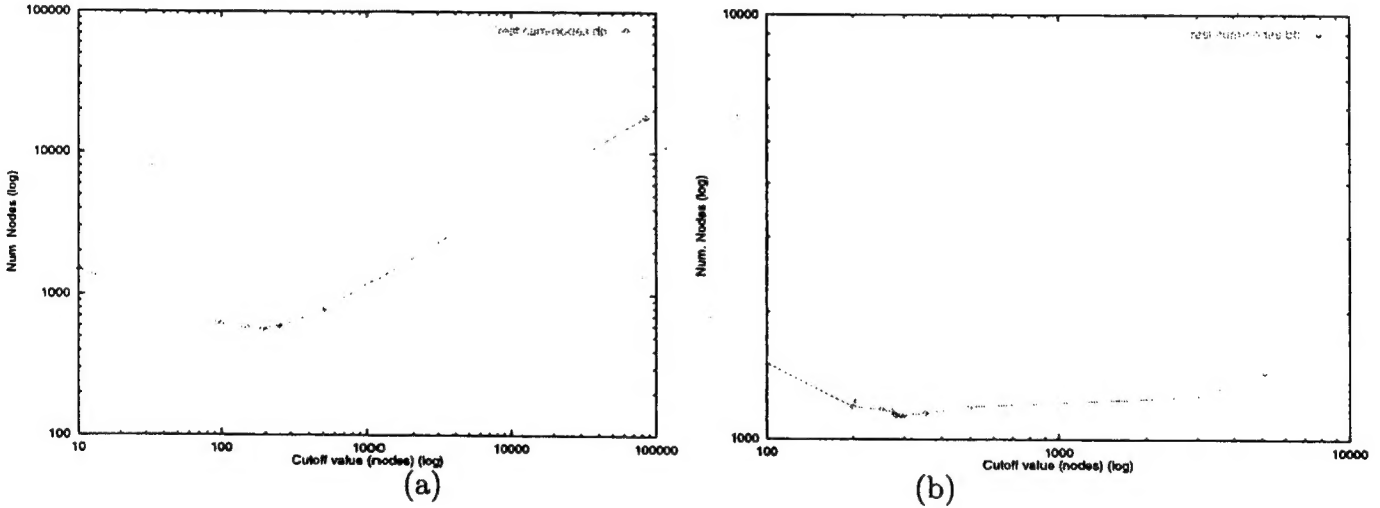

Figure 11: Expected cost as a function of restart cutoff for (a) depth-first and (b) branch-and-bound.

the strong heavy-tailed nature that characterizes depth-first search. In fact, that is also the reason why, as we increase the number of processors, it is worth running depth-first search on more processors. Intuitively, the chance of a short run with depth-first increases considerably.

7 Conclusions

We have provided results showing the computational advantage of a portfolio approach for dealing with hard combinatorial search and reasoning problems. Our results considered two predominant representation paradigms for combinatorial problems: Constraint Satisfaction formulations and Mixed Integer Programming formulations. Our analysis shows that one can exploit the large variance in certain randomized search methods by running them in a portfolio strategy and obtaining a superior overall performance, compared to more conservative algorithmic strategies. The portfolio approach appears particularly well-suited for the rapidly emerging compute-cluster paradigm. As our experiments show, the portfolio approach suggests new randomized search strategies. In particular, there is an advantage in optimizing the chance of finding a solution early on in a run, even though, on a single processor, this may lead to a larger overall expected runtime than that of other more traditional search techniques. Finally, if only a single processor is available, random restarts of a stochastic method is often the optimal strategy. These results suggest that ideas developed in the flexible computation community can play a significant role in practical algorithm design.

References

Aldous, D. and Vazirani, U. (1994). *Proc. of the 35th Symp. on the Found. of Comp. Sci.*, IEEE Press (1994) 492–501.

Alt, H., Guibas, L., Mehlhorn, K., Karp, R., and Wigderson A. (1996) A method for obtaining randomized algorithms with small tail probabilities. *Algorithmica*, 16, 1996, 543–547.

Anderson, L. (1985). Completing Partial Latin Squares. *Mathematisk Fysiske Meddelelser*, 41, 1985, 23–69.

Bixby, R.E. , Ceria, C.M., McZeal, C.M., Savelsbergh, M.W.P. (1996) An updated mixed integer programming library: MIPLIB 3.0. *SIAM News*, 1996.

Brelaz, D. (1979). New methods to color the verices of a graph. *Comm. of the ACM* (1979) 251–256.

Cheeseman, Peter and Kanefsky, Bob and Taylor, William M. (1991). Where the Really Hard Problems Are. *Proceedings IJCAI-91*, 1991, 163–169.

Colbourn, C. (1983). Embedding Partial Steiner Triple Systems is NP-Complete. *J. Combin. Theory* (A) 35 (1983), 100-105.

Dean, T. and Boddy, M. (1988) An analysis of time-dependent planning. *Proc. AAAI-88*, St. Paul, MI (1988) 49–54.

Dechter, R. (1991) Constraint networks. *Encyclopedia of Artificial Intelligence* John Wiley, New York (1991) 276-285.

Denes, J. and Keedwell, A. (1974) Latin Squares and their Applications. *Akademiai Kiado, Budapest, and English Universities Press*, London, 1974.

Ertel, W. (1991) Performance analysis of competitive or-parallel theorem proving. University of Munchen, Techn. report FKI-162-91, 1992.

Freuder, E. and Mackworth, A. (Eds.). *Constraint-based reasoning.* MIT Press, Cambridge, MA. USA, 1994.

Frost, D. . Rish, I., and Vila. L. (1997) Summarizing CSP hardness with continuous probability distributions. *Proc. AAAI-97*. New Providence. RI, 1997, 327–333.

Fujita, M., Slaney, J., and Bennett. F. (1993). Automatic Generation of Some Results in Finite Algebra *Proc. IJCAI*, 1993.

Gent, I. and Walsh, T. (1996) The Satisfiability Constraint Gap. *Artificial Intelligence*, 81, 1996.

Gomes, C.P. and Selman, B. (1997) Problem structure in the presence of perturbations. *Proc. AAAI-97*, New Providence, RI, 1997, 221–226.

Gomes, C.P. and Selman, B. (1997) Algorithm Portfolio Design: Theory vs. Practice. In *Proc. UAI-97*, New Providence, RI, 1997.

Gomes, C.P., Selman, B., Crato, N. (1997) Heavy-Tailed Distributions for Combinatorial Search. *Proc. Constraint Programming*, Linz, Austria, Nov. 1997.

Horvitz, E. and Klein, A. (1995) Reasoning, metareasoning, and mathematical truth: studies of theorem proving under limited resources. *Proc. of the Eleventh Conference on Uncertainty in Artificial Intelligence (UAI-95)*. August 1995.

Horvitz, E. and Zilberstein S. (1996) (Eds.) *Proceedings of Flexible Computation*, AAAI Fall Symposium, Cambridge. MA. 1996.

Huberman, B.A., Lukose, R.M., and Hogg, T. (1997). An economics approach to hard computational problems. *Science*, 265, 51–54.

Hogg, T., Huberman, B.A., and Williams, C.P. (Eds.) (1996). Phase Transitions and Complexity. *Artificial Intelligence*, 81 (Spec. Issue; 1996)

Kirkpatrick, S. and Selman, B. (1994) Critical Behavior in the Satisfiability of Random Boolean Expressions. *Science*, 264 (May 1994) 1297–1301.

Lam, C., Thiel, L., and Swiercz, S. (1989) *Can. J. Math.*, Vol. XLI, 6, 1989, 1117-1123.

Luby, M., Sinclair A., and Zuckerman, D. (1993). Optimal speedup of Las Vegas algorithms. *Information Process. Lett.*, 17, 1993, 173–180.

Mitchell, D., Selman, B., and Levesque. H.J. (1989) Hard and easy distributions of SAT problems. *Proc. AAAI-92*. San Jose. CA (1992) 459-465.

Motwani, R. and Raghavan, P. (1995) *Randomized algorithms.* Cambridge University Press: Cambridge, England, 1995.

Puget, J-F., and Leconte, M. (1995). Beyond the Black Box: Constraints as objects. *Proceedings of ILPS'95*, MIT Press, 513–527.

Russell, S and Norvig P. (1995) *Artificial Intelligence a Modern Approach.* Prentice Hall, Englewood Cliffs, NJ. (1995).

Selman, B. and Kirkpatrick, S. (1996) Finite-Size Scaling of the Computational Cost of Systematic Search. *Artificial Intelligence.* Vol. 81, 1996, 273–295.

Smith, B. and Dyer, M. Locating the Phase Transition in Binary Constraint Satisfaction Problems. *Artificial Intelligence.* 81. 1996.

Trick, M. and Johnson, D. (Eds.) (1996) *Proc. DIMACS Challenge on Satisfiability Testing, Graph Coloring, and Cliques.* DIMACS Series on Discr. Math., Am. Math. Soc. Press (1996).

Williams, C.P. and Hogg, T. (1992) Using deep structure to locate hard problems. *Proc. AAAI-92*, San Jose, CA, July 1992. 472–277.

Zhang, W. and Korf, R. A Study of Complexity Transitions on the Asymmetric Travelling Salesman Problem. *Artificial Intelligence*, 81, 1996.